Helsingin yliopisto
Tietojenkäsittelytieteen laitos
Ohjelmistotuotantoprojekti

Esimerkkituoteperhe

Suunnitteludokumentti

03.05.2004
Ryhmä 6
Juha Andersson
Jarmo Kielosto
Leo Linnamaa
Jan Tilles
Joose Vettenranta

Versiohistoria

Versio	Päivämäärä	Muutokset
1.4	03.05.2004	Jäädytetty versio
1.3	28.04.2004	Liitettä 5 tarkennettu
1.2	22.04.2004	FTR:n jälkeiset muutokset
1.1	04.04.2004	Arkkitehtuuriluku ja testaussuunnitelma päivitetty
1.0	24.03.2004	Ensimmäinen, vaillinainen versio ryhmän ja asiakkaan kommentoitavaksi

SISÄLTÖ

1 JOHDANTO 4

1.1 DOKUMENTIN TARKOITUS 4
1.2 TERMIT JA LYHENTEET 4

2 OHJELMISTOPERHEEN RAKENNE 6

2.1 OHJELMISTOPERHEEN YLEISKUVAUS 6
2.1.1 Erikoistaminen ja komponenttien vaihtaminen 7
2.1.2 Komponenttien esittely ja peruskomponenttien väliset suhteet 8
2.1.3 Komponenttien erikoistaminen sovelluksissa 12
2.1.3.1 Mökkitikkasovelluksen erikoistamat komponentit 12
2.1.3.2 Snooker-sovelluksen erikoistamat komponentit 14
2.1.3.3 Keilailusovelluksen erikoistamat komponentit 16
2.2 CORE-KOMPONENTTI 18
2.2.1 Core 19
2.2.2 ComponentLoader 20
2.2.3 ComponentReference 20
2.3 APPLICATIONMANAGEMENT-KOMPONENTTI 22
2.3.1 ApplicationManager 23
2.3.2 ApplicationInitializer 23
2.3.3 ApplicationDeInitializer 24
2.3.4 DatabaseApplicationManagement-komponentti 24
2.3.4.1 DatabaseInitializer 24
2.3.4.2 DatabaseDeInitializer 25
2.4 GAMEMANAGEMENT-KOMPONENTTI 26
2.4.1 GameManager 26
2.4.2 GameEventHandler 28
2.4.3 DartsGameManagement-komponentti 29
2.4.3.1 DartsGameManager 29
2.4.3.2 DartsGameEventHandler 29
2.4.4. SnookerGameManagement-komponentti 29
2.4.4.1 SnookerGameManager 29
2.4.4.2 SnookerGameEventHandler 30
2.4.5 BowlingGameManagement-komponentti 30
2.4.5.1 BowlingGameManager 30
2.4.5.2 BowlingGameEventHandler 31
2.5 PLAYERMANAGEMENT-KOMPONENTTI 32
2.5.1 Player 33
2.5.2 PlayerManager 35
2.6 RESULTMANAGEMENT-KOMPONENTTI 36
2.6.1 DataHolder 37
2.6.2 ResultHolder 39
2.6.3 DataContainer 39
2.7 REPORTMANAGEMENT-KOMPONENTTI 40
2.7.1 ReportManager 41
2.7.2 FileHandling 42
2.7.3 ResultListing 43
2.7.4 HighScore 43
2.7.5 BowlingReport-komponentti 44
2.7.6 SnookerReport-komponentti 45
2.7.7 DatabaseReport-komponentti 45
2.8 LOGIC-KOMPONENTTI 47
2.8.1 GameLogic 49
2.8.2 GameState 50

2.8.3 GameValidator *50*
2.8.4 ScoreCounter *51*
2.8.5 SimpleLogic-komponentti *52*
2.8.5.1 SimpleLogic 52
2.8.5.2 SimpleState 52
2.8.5.3 SimpleValidator 52
2.8.5.4 SimpleHelper 53
2.8.6 SnookerLogic-komponentti *53*
2.8.7 BowlingLogic-komponentti *54*
2.8.7.1 BowlingState 54
2.8.7.2 BowlingValidator 54
2.8.7.3 BowlingCounter 55
2.9 GUI-KOMPONENTTI 57
2.9.1 GUI *58*
2.9.2 GameEventComponent *59*
2.9.3 SimpleGui-komponentti *60*
2.9.4 GameGUI *60*
2.9.5 GameGUIManager *60*
2.9.6 SimpleGUI:n käyttöliittymäkomponenttikirjasto *61*
2.9.7 SnookerGUI-komponentti *62*
2.9.8 DartsGui-komponentti *62*
2.9.9 Bowling-komponentti *62*
2.10 SEKVENSSIKAAVIOT 64
2.10.1 Ohjelman käynnistymisen sekvenssikaavio *64*
2.10.2 Pisteiden lisäyksen sekvenssikaavio *66*
2.10.3 Loppuraportin luomisen sekvenssikaavio *69*

3 KÄYTTÖLIITTYMIEN ULKOASU 71

3.1 MÖKKITIKKA 71
3.1.1 Mökkitikan menubar *72*
3.1.2 Mökkitikan loppuraportti *75*
3.2 SNOOKER 76
3.2.1 Snookerin menubar *76*
3.2.2 Snookerin loppuraportti *79*
3.3 KEILAILU 80
3.3.1 Keilailun menubar *81*
3.3.2 Keilailun loppuraportti *84*

4 TESTAUSSUUNNITELMA 85

4.1 TESTAUKSEN TAVOITTEET 85
4.2 TESTAUKSESSA KÄYTETTÄVÄT APUVÄLINEET JA TESTIYMPÄRISTÖ 85
4.3 TESTIVAIHEET 86
4.3.1 Yksikkötestaus *86*
4.3.2 Integrointitestaus *87*
4.3.3 Järjestelmätestaus *87*
4.3.4 Hyväksymistestaus *88*
4.4 TESTIVAATIMUSTEN VALIDOINTI 88

LIITE 1: ALUSTUSTIEDOSTON RAKENNE 90

LIITE 2: TAPAHTUMIEN INDEKSIT 91

LIITE 3: TULOSLISTAT 92

LIITE 4: JÄRJESTELMÄTESTAUKSEN TESTITAPAUKSET 93

LIITE 5: VAATIMUKSIIN VIITTAAMINEN ERI LUVUISSA 95

1 Johdanto

Projektin aikana toteutetaan yksinkertainen tuoteperhe Java-ohjelmointikielellä. Tuoteperheellä tarkoitetaan tässä joukkoa samankaltaisia sovelluksia, joilla on yhteinen kehysrakenne. Tämän projektin tuoteperhe sisältää kolme sovellusta, jotka toimivat esimerkkiaineistona RITA-projektissa. Sovellusten aihepiiriksi asiakas ehdotti harrastuspelien kirjanpito- ja pistelaskuohjelmia. Projektiryhmä toteuttaa seuraavat pelit: **1) mökkitikka, 2) snooker ja 3) keilailu**.

1.1 Dokumentin tarkoitus

Suunnitteludokumenttiin on koottu ohjelmistoperheen toteuttamiseen liittyvät päätökset. Dokumentti toimii projektiryhmän ohjeena toteutusvaiheen aikana: toteutus esitetään sillä tarkkuudella, että ohjelmointi on mahdollista toteuttaa suoraviivaisesti. Suunnittelussa on otettu huomioon vaatimusdokumentissa esitetyt asiakkaan toiminnalliset, laadulliset ja testivaatimukset.

Luvussa 2 esitetään ohjelmistoperheen rakenne komponenteittain. Rakenne kuvataan luokkien ja metodien tarkkuudella. Luvussa 3 kuvataan sovellusten käyttöliittymät sekä luvussa 4 esitetään testaussuunnitelma.

1.2 Termit ja lyhenteet

Isäkomponentti
Isäkomponentti viittaa komponenttiin, joka on erikoistettua komponenttia korkeammalla tasolla arkkitehtuurissa eli se komponentti, josta kyseinen komponentti on erikoistettu. Isäkomponentti voi olla peruskomponentti tai erikoistettu komponentti.

Korkeampi taso

Arkkitehtuurissa korkeammalla tasolla tarkoitetaan kerrosta, joka on lähempänä peruskomponentteja. Peruskomponenttikerros on korkein taso.

Matalampi taso

Arkkitehtuurissa matalammalla tasolla tarkoitetaan kerrosta, joka on peruskomponenteista laskien kauempana kuin nyt kyseessä oleva kerros.

Peruskomponentti

Peruskomponentilla tarkoitetaan kehysrakenteen korkeimman tason komponenttia, jota ei ole erikoistettu.

2 Ohjelmistoperheen rakenne

Tämä luku esittelee ohjelmistoperheen yhteistä kehysrakennetta ja kuvaa, mitä kohtia projektin aikana tuotettavat sovellukset kehysrakenteesta erikoistavat omaan käyttöönsä. Luku sisältää kuvauksen kehysrakenteeseen kuuluvien komponenttien välisistä suhteista. Sovelluskohtaisesti esitellään ytimen erikoistamiskohdat sekä lopuksi kerrotaan komponenttien sisäiset rakenteet, luokkien väliset suhteet ja kehysrakenteen toiminnan osalta tärkeimmät metodit. Erikoistettujen luokkien yhteydessä on kuvattu ne metodit jotka ovat ko. luokasta erikoistettu. Tässä dokumentissa viitataan vaatimusdokumentissa numeroituihin vaatimuksiin, kun halutaan osoittaa, miten kyseinen vaatimus toteutetaan. Viittaukset on koottu liitteeseen 5.

2.1 Ohjelmistoperheen yleiskuvaus

Ohjelmistoperheen kehysrakenne jakaantuu kahdeksaan peruskomponenttiin, joista sovellukset voivat erikoistaa käyttöönsä omia komponenttejaan. Projektissa toteutettavat kolme sovellusta sisältävät yhteensä 11 erikoistettua komponenttia (vaatimus 2.3.14). Komponenttirakenne löytyy kuvasta 2.2. Komponenteista kaksi toteutetaan tyhjinä runkoina, mikä tarkoittaa sitä, että niiden erikoistamiskohdat ja metodit kerrotaan, mutta metodit eivät sisällä ohjelmakoodia

.

Kun peruskomponentista erikoistetaan uusi komponentti, voidaan uuteen komponenttiin lisätä luokkia ja metodeja sovelluksen toteutuksen yhteydessä. Jos komponenttia erikoistetaan edelleen uudessa komponentissa, perustuu laajennus tähän komponenttiin eikä peruskomponenttiin. Näin saadaan komponenttien välille kolmitasoinen arkkitehtuuri. Ensimmäiseksi tasoksi kutsutaan tasoa, jossa sijaitsevat peruskomponentit, ja kaikki näistä erikoistetut komponentit sijaitsevat tasoilla kaksi ja kolme (vaatimus 2.3.7).

Komponenttien rakennetta on painotettu siten, että mahdollisimman paljon toteutuksesta olisi peruskomponentissa ja mahdollisimman vähän jouduttaisiin erikoistamaan. Lähdekoodin painotuksella peruskomponentteihin pyritään maksimoimaan koodin mahdollisimman suuri kierrätettävyys sovellusten välillä (vaatimus 2.3.8). Kuitenkin kahden komponentin osalta koodin painottuminen peruskomponentteihin ei ole mahdollista, vaan lähdekoodin määrä painottuu selkeästi arkkitehtuurin toiselle ja kolmannelle tasolle. Nämä kaksi peruskomponenttia ovat GUI ja GameManagement. GUI:ssa tämä johtuu siitä, että eri käyttöliittymät asettelevat tekstikentät ja muut käyttöliittymäkomponentit eri tavoin. Sama syy heijastuu GameManagement-komponenttiin, mihin GUI:sta tulevat tapahtumat ohjataan. Tapahtumiin reagoidaan eri tavoin pelin luonteen mukaisesti.

Peruskomponenttien toteutus pyrkii noudattelemaan mahdollisimman paljon mökkitikkasovelluksen tarvitsemia ratkaisuja. Tällöin mökkitikkasovellus tulee olemaan sovelluksista vähiten erikoistettava (vaatimus 2.3.2). Peruskomponenttien sisältämät rajapinnat on kuitenkin valittu niin yleisellä tasolla, että kaikki sovellukset voidaan toteuttaa helposti erikoistamalla komponentteja sovelluksien tarpeisiin sopiviksi. Tällöin ydintä ei tarvitse erikseen muokata eri sovelluksia varten, vaan riittää, että sovelluksen erikoistamiskohdat esitellään sovelluksen käynnistämisen yhteydessä (vaatimus 2.3.5).

2.1.1 Erikoistaminen ja komponenttien vaihtaminen

Mikäli erikoistamiskohtia lisätään myöhemmin kehysrakenteeseen, joudutaan myös päivittämään ydintä. Sovelluksien erikoistamiskohdat kerrotaan ytimelle erillisen tiedoston avulla, jossa kuvataan, mitkä komponenttien sisältämät luokat on erikoistettu ja mitkä ovat erikoistettujen luokkien nimet. Tiedoston rakenne on kuvattu tarkemmin liitteessä 1. Koska erikoistetut luokat voivat laajentaa peruskomponenttien toteutusta eri tavoin, ei voida taata, että erikoistetuista komponenteista edelleen erikoistetut komponentit olisivat rajapinnaltaan keskenään yhteneviä. Tällöin voidaan vaihtaa erikoistettuja komponentteja sovellusten kesken, esimerkiksi vaihdetaan kahden pelin Logic-komponenttia keskenään (vaatimus 2.3.3). Tästä seuraa myös, että

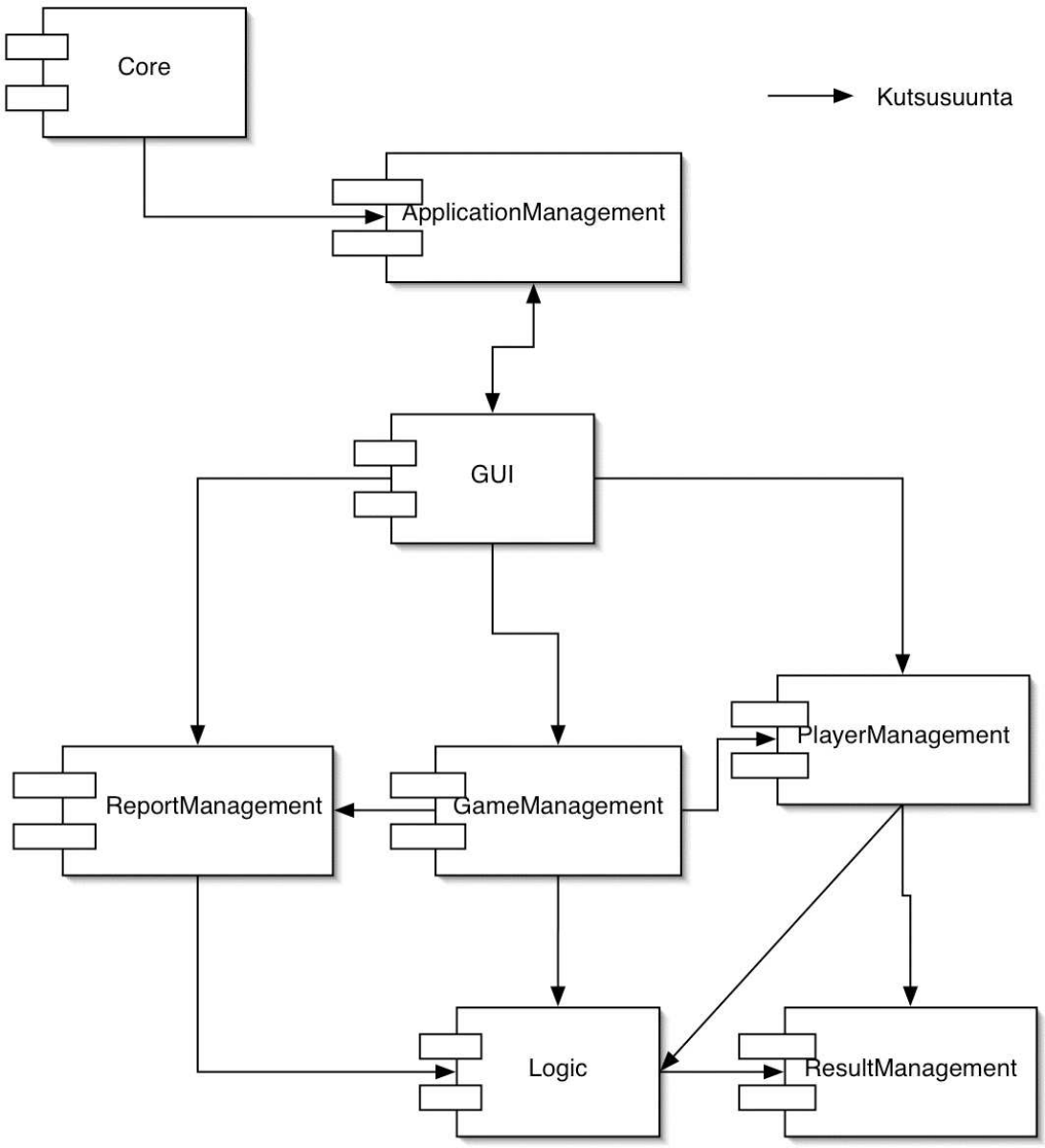
vaihdettavien komponenttien tulee sijaita keskenään samalla arkkitehtuurin tasolla. Kun komponentteja vaihdetaan sovellusten kesken, seuraa epävakautta ohjelmiston erikoistamiskohtien yhteydessä, eikä tällöin saatu uusi sovellus ei ole välttämättä ajonaikaisesti vakaa, mutta se on kuitenkin käännettävissä.

Jos uudet sovellukset käyttävät samoja erikoistamiskohtia, ei päivityksiä tarvitse tehdä (vaatimus 2.2.2). Tarpeelliset päivitykset liittyvät Core-komponentin ComponentRefence-luokkaan, joka hallinnoi kehysrakenteen erikoistamiskohtia kaikilla arkkitehtuurin tasoilla.

Ohjelmistoperheen komponenttien nimissä on käytetty yhtenäisiä nimeämistapoja erikoistettujen luokkien ja komponenttien yhteydessä. Luokkien nimistä voi päätellä, mihin erikoistettuun komponenttiin kyseinen luokka kuuluu (vaatimus 2.3.1): esimerkiksi DartsGuiManager kuuluu komponenttiin DartsGui. Myös komponenttien välillä on yhteneviä nimeämisratkaisuita. Kuvissa on käytetty komponenteista standardia komponenttien esitystapaa. Erikoistamiskohdat ja suunnat on kuviin merkitty yhtenevästi samankaltaisilla nuolilla.

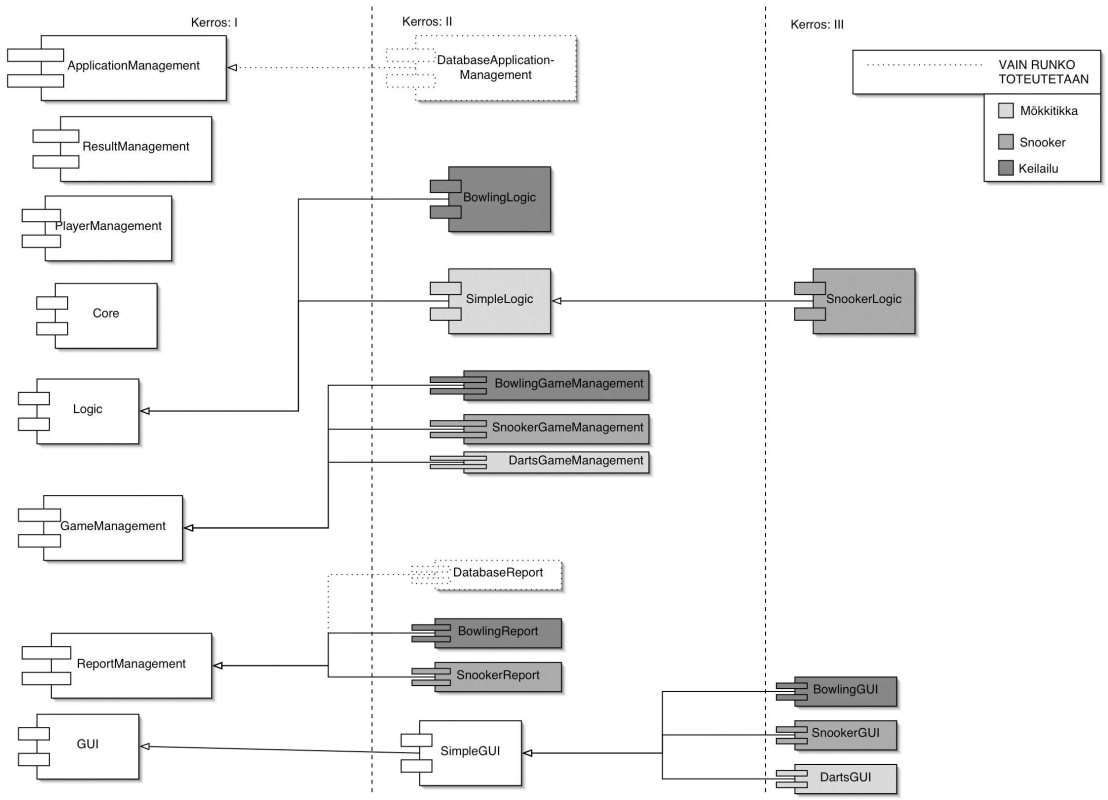
2.1.2 Komponenttien esittely ja peruskomponenttien väliset suhteet

Peruskomponentteja on ytimessä yhteensä kahdeksan kappaletta. Niiden keskinäiset suhteet on esitetty kuvassa 2.1. Core- ja ApplicationManagement-komponentit huolehtivat sovelluksen käynnistymisestä ja hallitusta sulkeutumisesta. Varsinainen pelin ohjaus on toteutettu käyttöliittymässä tapahtumaohjatusti. GUI-komponentissa tapahtumakuuntelijat reagoivat käyttäjän toimiin ja kuuntelijat ohjaavat tapahtumat muille komponenteille. GameManagement-komponenttiin on pyritty keskittämään pelinaikaisiin tapahtumiin liittyvien tilanteiden hallinta. ReportManagement-komponentti sisältää toiminnot tuloslistojen hallinnointiin ja ulkoasun muokkaukseen. PlayerManagement-komponentti keskittyy pelaajatietojen hallintaan. Logic-komponentti sisältää pelin logiikkaan, syötteisiin ja pisteenlaskuun liittyviä määreitä, rajoituksia ja toteutuksia. ResultManagement-komponentti huolehtii pelin aikaisesta pisteiden tallentamisesta ja hakemisesta.



Kuva 2.1: Yleinen komponenttikaavio

Sovellusten käyttämät erikoistetut komponentit on kuvattu kuvassa 2.2. Kuvasta käy ilmi peruskomponenteista sovelluskohtaisesti erikoistetut komponentit (erotettu väreillä), sekä kehyksen kerrosrakenteen rajat. Kuvasta näkyy että eri sovellukset käyttävät osittain samoja peruskomponentteja tai erikoistavat niitä sovelluskohtaisesti niin että, joitakin komponentteja erikoistavat kaikki sovellukset, joitakin vain kaksi tai yksi sovellusta (vaatimus 2.3.15).

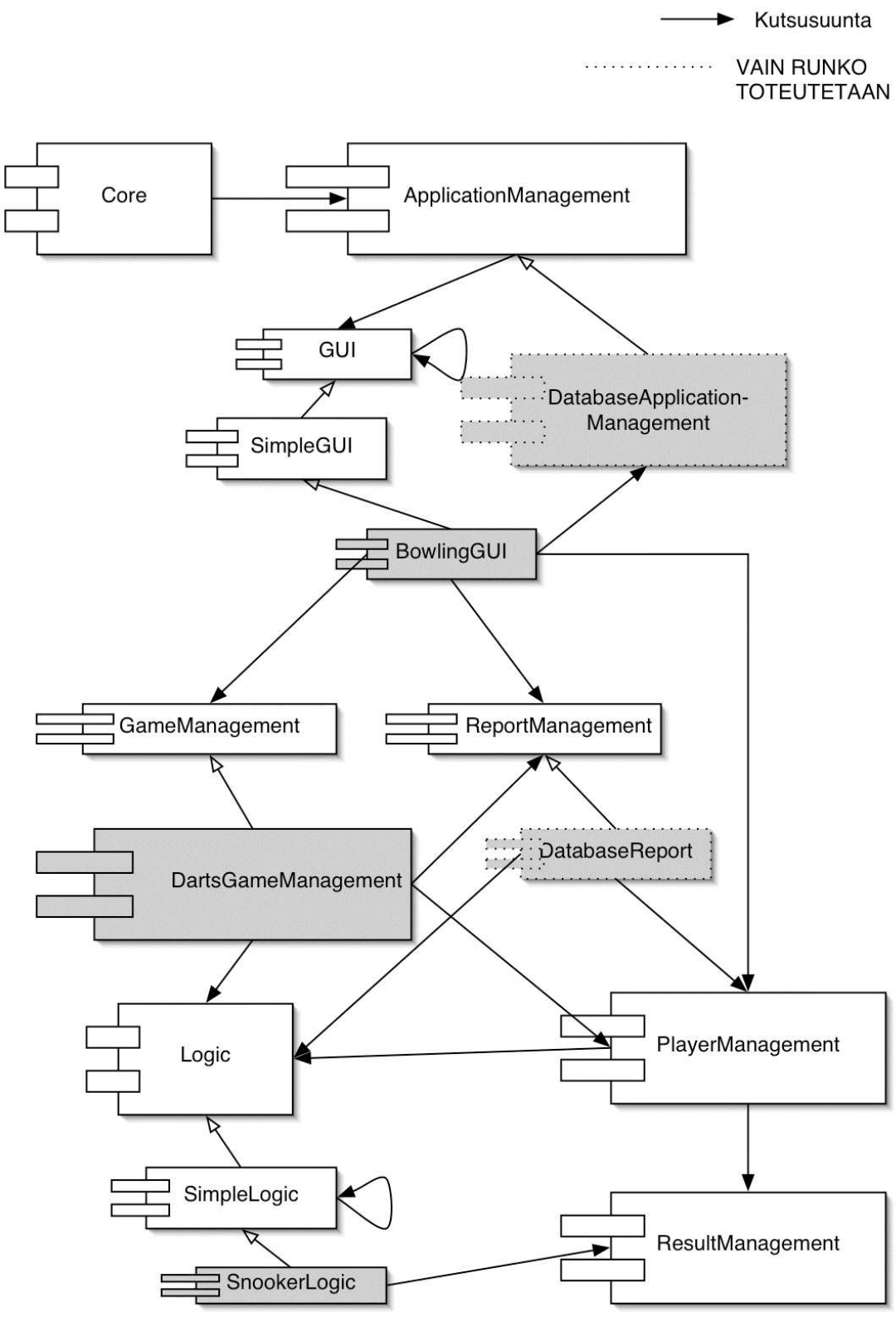


Kuva 2.2: Erikoistetut komponentit

Kuva 2.2 eroaa vaatimusdokumentin kuvasta 4.1. seuraavien komponenttien osalta:

- Kuvassa 4.1 Report-komponentti on nimetty uudelleen ReportManagement-komponentiksi.
- Kuvan 4.1 BowlingResult on jätetty pois kokonaan, suunnitteluvaiheessa kävi ilmi että komponentti on tarpeeton.
- Suunnitteluvaiheessa poistettiin Keilailu-sovellusta varten vaatimusdokumentin määrittelyssä lisätty BowlingPlayer joka erikoisti PlayerManagement-komponenttia.
- Lisäksi ReportManagement-komponenttia erikoistaviin komponentteihin on lisätty BowlingReport- ja SnookerReport-komponentit.

Rungosta voidaan toteuttaa sovelluksia myös komponentteja mielivaltaisesti yhdistelemällä. Kuvassa 2.3 on esitetty tällainen ”haamusovellus”, jossa käytetään komponentteja, joiden sisällä metodit eivät sisällä ohjelmakoodia ts. ne ovat tyhjiä komponentteja (vaatimus 2.3.11). Haamusovellukselle on luotava samanlainen käynnistystiedosto kuten mille tahansa muulle ydintä käyttävälle sovellukselle.



Kuva 2.3: Komponenttikaavio ”haamusovelluksesta”

2.1.3 Komponenttien erikoistaminen sovelluksissa

Tässä luvussa esitellään kunkin sovelluksen erikoistamat komponentit.

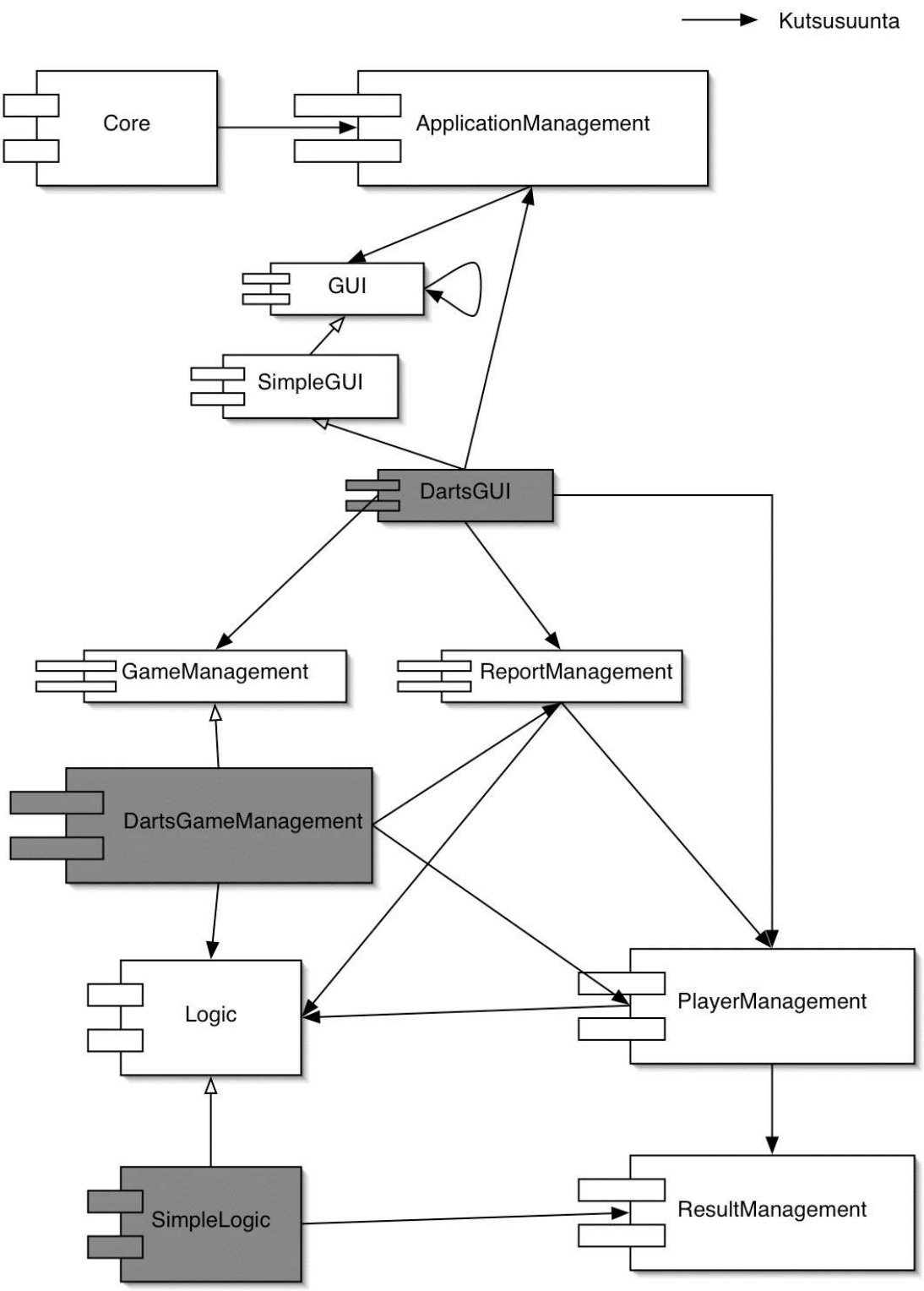
2.1.3.1 Mökkitikkasovelluksen erikoistamat komponentit

Mökkitikkasovellus erikoistaa peruskomponenteista kolme komponenttia. Nämä komponentit ovat:

- DartsGui, joka erikoistetaan SimpleGUI-komponentista, joka puolestaan erikoistaa GUI-komponenttia,
- DartsGameManagement, joka erikoistetaan GameManagement-komponentista, ja
- SimpleLogic, joka erikoistetaan Logic-komponentista.

DartsGui-komponentin erikoistaminen tapahtuu kolmitasoisesti. GUI-komponentti erikoistetaan ensin SimpleGUI-komponentista, josta erikoistetaan DartsGUI. Komponentti erikoistetaan, jotta saavutetaan sovelluskohtainen käyttöliittymän ulkoasu.

DartsGameManagement on erikoistettu suoraan GameManagement-komponentista. Se sisältää pelitapahtumien käsittelijät ja pelikohtaisten toimintojen toteutuksia. Pelikohtaisten toimintojen toteutuksella tarkoitetaan esimerkiksi mitä sovelluksessa tapahtuu, kun syötetään vääränlainen syöte kenttään tai pelaaja haluaa aloittaa uuden pelin kesken käynnissä olevan pelin. SimpleLogic on erikoistettu Logic-komponentista. SimpleLogic toteuttaa mökkitikan logiikan.



Kuva 2.4: Mökkitikkasovelluksen komponenttikaavio

2.1.3.2 Snooker-sovelluksen erikoistamat komponentit

Snooker-sovellus erikoistaa peruskomponenteista neljä komponenttia. Nämä komponentit ovat:

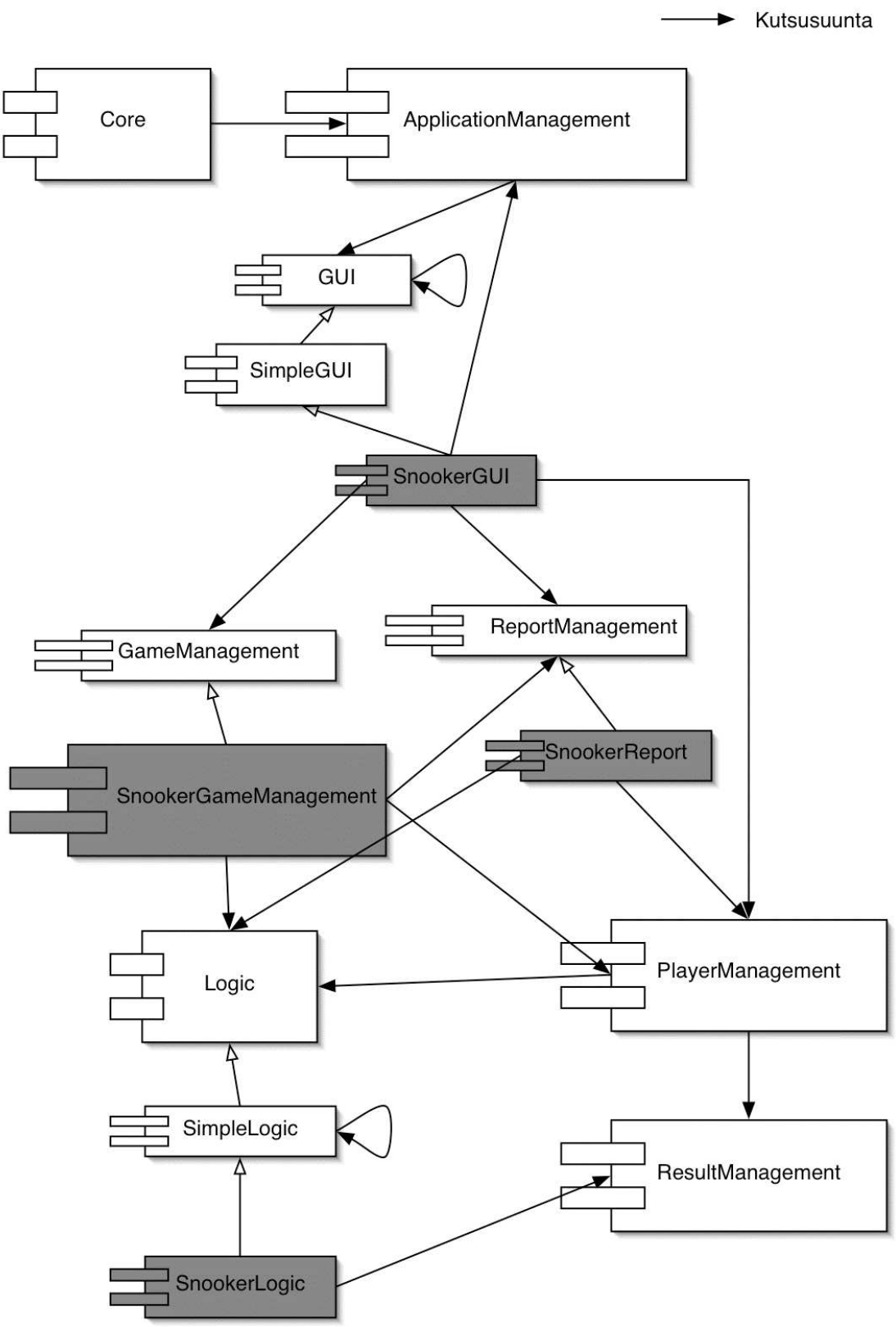
- SnookerGui, joka erikoistetaan SimpleGUI -komponentista, joka puolestaan erikoistetaan GUI komponentista,
- SnookerGameManagement, joka erikoistetaan GameManagement-komponentista,
- SnookerLogic, joka erikoistetaan SimpleLogic komponentista, joka puolestaan erikoistetaan Logic-komponentista, ja
- SnookerReport, joka erikoistetaan Report-komponentista.

SnookerGui-komponentin erikoistaminen tapahtuu kolmitasoisesti. GUI-komponentti erikoistetaan ensin SimpleGUI-komponentista, josta erikoistetaan SnookerGUI. Komponentti erikoistetaan, jotta saavutetaan sovelluskohtainen käyttöliittymän ulkoasu.

SnookerGameManagement on erikoistettu suoraan GameManagement-komponentista. Se sisältää pelitapahtumien käsittelijät ja pelikohtaisten toimintojen toteutuksia. Esimerkiksi Snooker-sovelluksessa voidaan syöttää miinuspisteitä. Kyseiset miinuspisteet lisätään vastapalaajan pisteisiin.

SnookerLogic on erikoistettu SimpleLogic-komponentista, joka puolestaan on erikoistettu Logic-komponentista. Komponentissa on siis kolmitasoinen rakenne (vaatimus 2.3.9). SimpleLogic komponentti toteuttaa Mökkitikan logiikan, sitä ei voida käyttää suoraan koska Snookerin pelilogiikka poikkeaa Mökkitikasta, siksi Snooker tarvitsee oman logiikka komponenttinsa.

SnookerReport-komponentti on erikoistettu Report-komponentista. Erikoistamisella toteutetaan erilainen HighScore-toiminto sovelluksen tarpeisiin.



Kuva 2.5: Snookerin komponenttikaavio

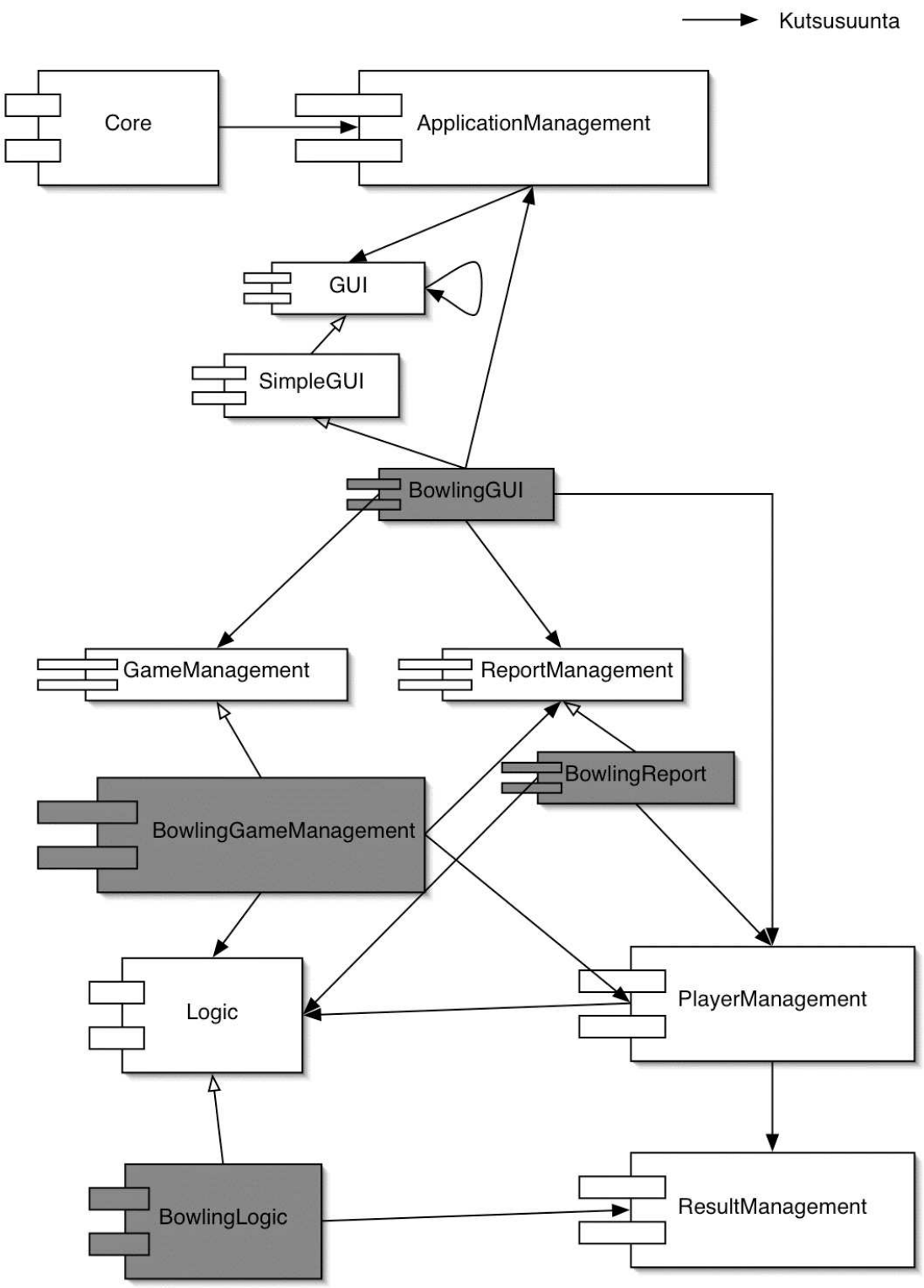
2.1.3.3 Keilailusovelluksen erikoistamat komponentit

Keilailusovellus erikoistaa peruskomponenteista neljä komponenttia. Nämä komponentit ovat:

- BowlingGui, joka erikoistetaan SimpleGUI -komponentista, joka puolestaan erikoistetaan GUI-komponentista
- BowlingGameManagement, joka erikoistetaan GameManagement-komponentista,
- BowlingLogic, joka erikoistetaan Logic-komponentista, ja
- BowlingReport, joka erikoistetaan Report-komponentista.

BowlingGui-komponentin erikoistaminen tapahtuu kolmitasoisesti. GUI-komponentti erikoistetaan ensin SimpleGUI-komponentista, josta erikoistetaan BowlingGUI. Komponentti erikoistetaan, jotta saavutetaan sovelluskohtainen käyttöliittymän ulkoasu.

BowlingGameManagement on erikoistettu suoraan GameManagement-komponentista. Komponentti sisältää pelitapahtumien käsittelijät ja pelikohtaisten toimintojen toteutuksia. Esimerkiksi Bowling-sovelluksessa on kokoajan pidettävä kirjaa välituloksista ja pelaajan tekemistä kaadoista ja paikoista, niin että pisteet voidaan laskea kierroskohtaisesti oikein. BowlingLogic on erikoistettu Logic-komponentista, niin että saadaan keilailun logiikka vastaava komponentti. BowlingReport on erikoistettu Report-komponentista. Erikoistamisella luodaan sovellukselle oma tuloslistan ulkoasu.



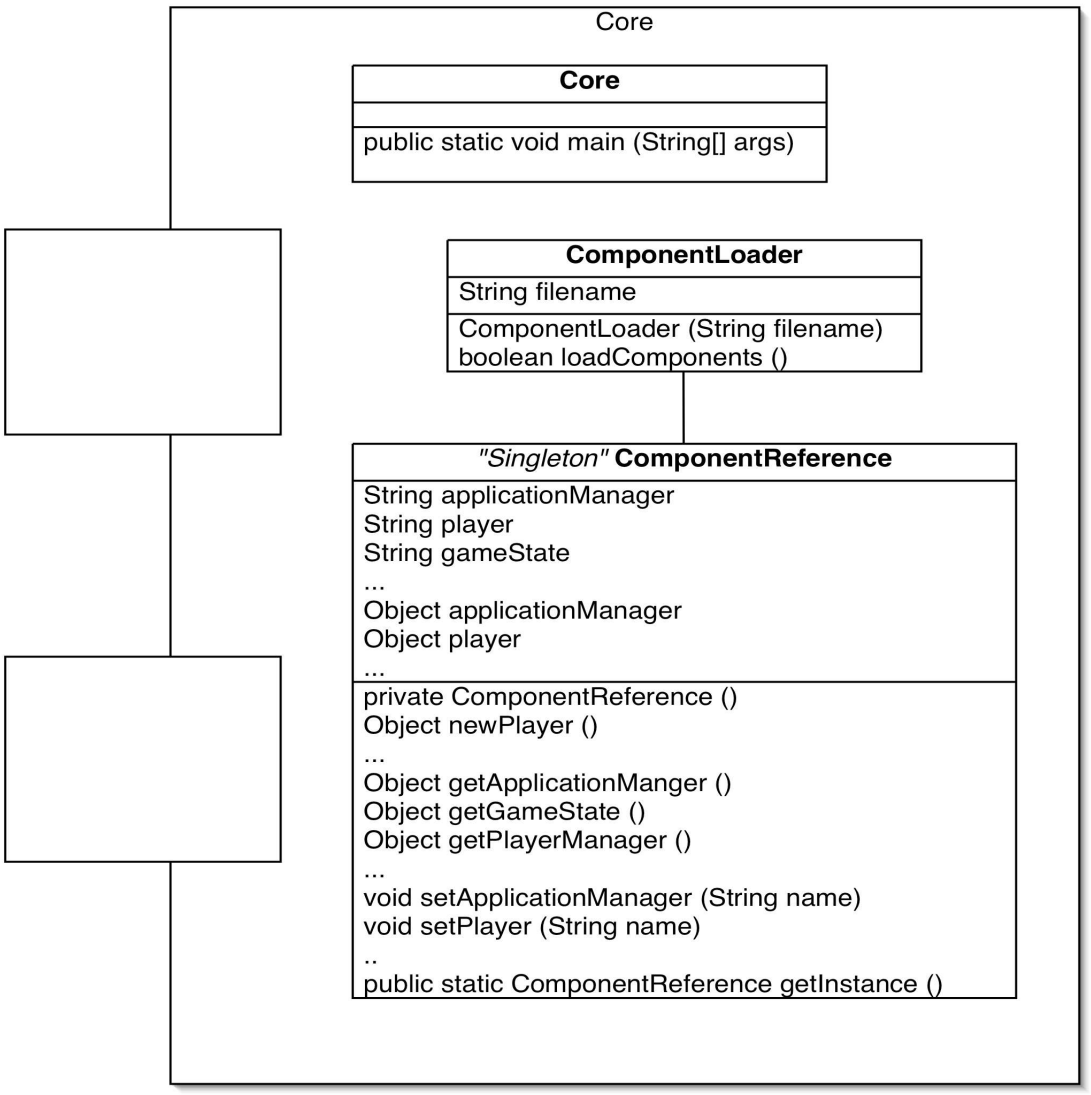
Kuva 2.6: Keilailun komponenttikaavio

Seuraavasta luvusta 2.2 eteenpäin esitellään komponenttien kehysrakenteen kannalta olennaiset metodit ja kuvataan komponenttien toimintaa tarkemmin.

2.2 Core-komponentti

Core-komponentin tehtävänä on huolehtia sovellusten yhtenäisestä käynnistymistavasta. Sovellukset eroavat toisistaan main()-metodille annettavan parametrin avulla. Parametri sisältää tiedostonimen ilman mitään tiedostopäätettä tai hakemistopolkua, tiedoston Core puolestaan antaa ComponentLoader-luokan luettavaksi. Parametrina annettu tiedosto sisältää tiedon niistä luokista, joita käynnistyvä sovellus erikoistaa. Alustustiedoston rakenne löytyy dokumentin liitteenä (Liite 1).

Kun erikoistamiskohdat ovat selvillä, Core kutsuu sovelluksen käyttämän ApplicationManagement-komponentin Initialize-metodia hyödyntäen aikaisemmin ComponentLoaderin luomaa ComponentReference-luokkaa (vaatimus 2.3.6). ComponentReference-luokan toteuttaa Singleton (ainokainen) suunnittelumallin ja on sovelluskohtainen luokka jonka tehtävänä on antaa sovellukselle viitteitä sen tarvitsemiin luokkiin tai luoda niitä tarvittaessa.



Kuva 2.7: Core-komponentti

2.2.1 Core

Sovellus käynnistetään tämän luokan avulla.

```
void main(String[] args)
```

Metodi saa parametrinaan tiedoston nimen, jossa määritellään ajettavan sovelluksen erikoistamiskohdat. Muut mahdollisesti annettavat parametrit jätetään huomioimatta. Kaikki sovellukset aloittavat suorituksensa tämän metodin kautta.

2.2.2 ComponentLoader

ComponentLoader-luokan tehtävänä on lukea annetusta tiedostosta ne luokan nimet muistiin, joita ajossa oleva sovellus erikoistaa, sekä erikoistettavien luokkien nimet. Nämä tiedot tallennetaan ComponentReference-luokkaan.

`ComponentLoader(String filename)`
Konstruktori saa parametrinaan tiedostonimen, jossa sovelluksen erikoistamiskohdat on kerrottu. Jos parametrina on null-arvo, konstruktori heittää poikkeuksen.

`boolean loadComponents()`
Metodi lukee tiedostosta viitteet erikoistamiskohdille ja tallentaa ne ComponentReference-olioon. Metodi palauttaa arvon false jos komponenttien lataamisessa tapahtuu virhe tai kyseisen pelin tiedostoa ei ole tai sitä ei voida lukea. Muussa tapauksessa metodi palauttaa ”true”.

2.2.3 ComponentReference

ComponentReference-luokka ylläpitää koko sovelluksen ajon ajan tietoja siitä, mitä luokkia sovellus erikoistaa. Sovelluksen erikoistamat komponentit käyvät ilmi niiden alustustiedostoista (Liite 1). Kaikki kehysrakenteeseen viittaavat luokat tulee luoda tämän luokan sisältämien metodien avulla. Samoin viitteet keskeisiin kehysrakenteessa oleviin luokkiin tulee hakea tämän luokan kautta. Luokka toteuttaa suunnittelumallin ainokainen (singleton).

`ComponentReference getInstance()`
Metodi paluttaa viitteen ainokaiseen (Singleton), joka siis on tämän luokan ainoa ajonaikainen ilmentymä. Sovelluksen käyttämistä muista luokista voi olla useampia ajonaikaisia ilmentymiä.

Luokassa on metodi kaikille peruskomponenttien luokille. Tällainen metodi palauttaa paluuarvonaan viitteen kyseiseseen luokkaan. Mikäli viitettä ei ole, luokka tekee

uuden ilmentymän. Lisäksi muutamien luokkien tapauksissa luokkaan kautta voidaan myös luoda peruskomponentin luokasta uusi ilmentymä. Tällöin metodi palauttaa viitteen uuteen luotuun luokkaan. Luokassa on myös set-metodit jokaista erikoistettavaa luokannimeä varten. Tällöin annetaan parametrina merkkijono.

Koska set- ja get-metodien nimet ovat kuvaavat, ne on pelkästään mainittu taulukossa 2.2.3. Osasta luokista on ajon aikana tarve luoda useampi kuin yksi ilmentymä. Näiden luokkien osalta get-metodit on nimetty new-metodiksi. Tällöin metodi palauttaa arvonaan uuden ilmentymän kyseisestä luokasta eikä viitettä vanhaan, kuten get-metodi.

Luokan hakemisen metodit	**Luokan nimen asettamisen metodit**
ApplicationManager getApplicationManager()	void setApplicationManager(String text)
ApplicationInitializer getApplicationInitializer()	void setApplicationInitializer(String text)
ApplicationDeInitializer getApplicationDeInitializer()	void setApplicationDeInitializer(String text)
GameManager getGameManager()	void setGameManager(String text)
GameEventHandler getGameEventHandler()	void setGameEventHandler(String text)
Player newPlayer()	void setPlayer(String text)
PlayerManager getPlayerManager()	void setPlayerManager(String text)
DataHolder newDataHolder()	void setDataHolder(String text)
DataContainer newDataContainer()	void setDataContainer(String text)
FileHandling getFileHandling()	void setFileHandling(String text)
ResultListing getResultListing()	void setResultListing(String text)
HighScore getHighScore()	void setHighScore(String text)
ReportManager getReportManager()	void setReportManager(String text)
GameLogic getGameLogic()	void setGameLogic(String text)
GameState getGameState()	void setGameState(String text)
GameValidator getGameValidator()	void setGameValidator(String text)
ScoreCounter newScoreCounter()	void setScoreCounter(String text)
SimpleHelper newSimpleHelper()	void setSimpleHelper(String text)
GUI getGui()	void setGui(String text)
GameGui getGameGuiManager()	void setGameGuiManager(String text)

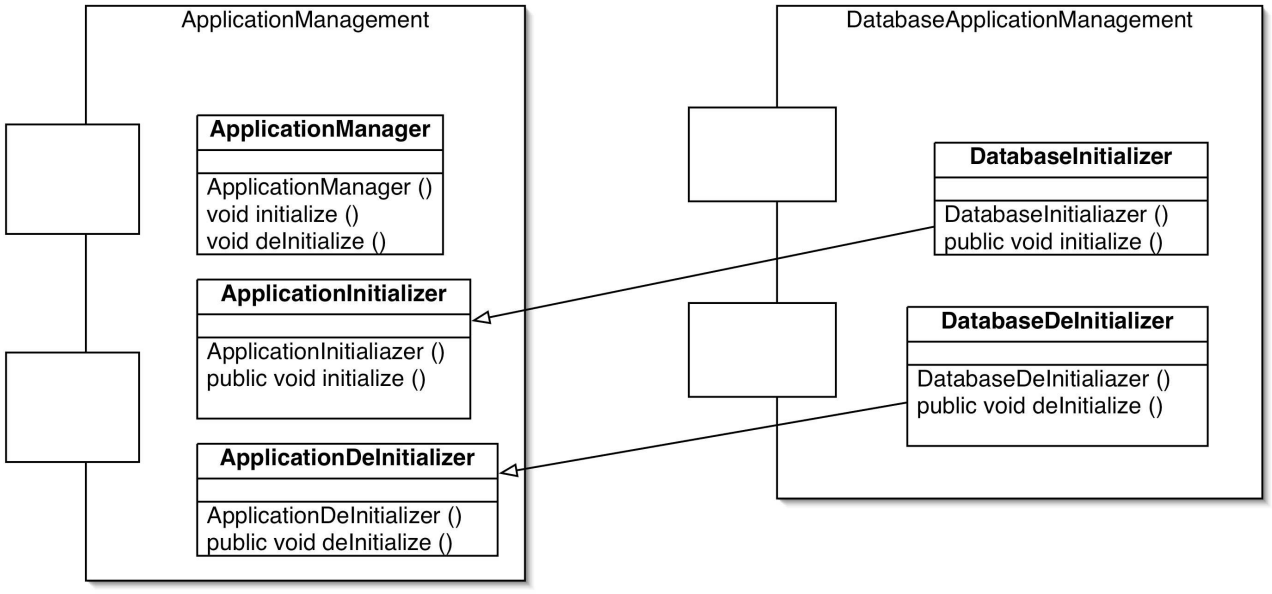
Taulukko 2.2.3: ComponentReference-luokan metodit

2.3 ApplicationManagement-komponentti

Komponentti keskittyy suorittamaan ne sovelluksen alustustoiminnot ja lopetustoiminnot, jotka varsinainen pisteenlaskusovellus tarvitse toimiakseen. Lisäksi komponentti huolehtia myös varsinaisen pelin käynnistymisestä, mikä tapahtuu käynnistämällä GUI.

ApplicationInitializer-luokan tehtävänä on käynnistää sovellusten alustukset ja luoda lopuksi käyttöliittymän.

ApplicationDeInitalizer-luokka huolehtii puolestaan sovelluksen sulkemisesta.



Kuva 2.8: ApplicationManagement-komponentti

2.3.1 ApplicationManager

Luokka ApplicationManager sisältää toteutuksen sovelluksen käynnistymiselle (initialization) ja sulkemiselle (deinitialization). Nämä kohdat voi kukin sovellus halutessaan erikoistaa.

`ApplicationManager()`
Konstruktori luokalle.

`void initialize()`
Metodia kutsuu Core-komponentin Core-luokka. Metodissa kutsutaan ApplicationInitializer-luokan initialize()-metodia, jossa suoritetaan sovelluksen tarvitsemat alustukset. Tämän jälkeen metodi käynnistää GUI:n ja lopuksi metodi jää odottamaan signaalia GUI-componentilta, että se on suljettu. Signaalin saatua metodi loppuu.

`void deInitialize()`
Metodia kutsutaan Core-komponentista, kun sovellusta ollaan sulkemassa. Metodi kutsuu ApplicationDeinitializer.deInitialize()-metodia, jossa voidaan toteuttaa mahdolliset sovellusten sisältämät sulkeutumistoimenpiteet. Lisäksi metodissa toteutetaan mahdolliset arkkitehtuurin vaatimat sulkemistoimenpiteet. Jos sovelluksessa on toteutettu tietokantayhteyksiä tai verkkoyhteyksiä, kyseiset yhteydet suljetaan samalla.

2.3.2 ApplicationInitializer

Luokka ApplicationInitializer on ApplicationManagement-komponentin erikoistamiskohta. Luokassa toteutetaan mahdolliset sovelluksen vaatimat alustus-toimenpiteet.

`ApplicationInitializer()`
Konstruktori luokalle.

`void initialize()`

Metodissa suoritetaan mahdolliset sovelluksen vaatimat alustustoimenpiteet.

2.3.3 ApplicationDeInitializer

Luokka ApplicationDeinitializer on ApplicationManagement-komponentin erikoistamiskohta. Luokassa toteutetaan sovelluksen sulkemisessa tarvittavat toimenpiteet.

`ApplicationDeInitializer()`

Konstruktori luokalle.

`void deInitialize()`

Metodissa toteutetaan sovelluksen sulkemisessa tarvittavat toimenpiteet.

2.3.4 DatabaseApplicationManagement-komponentti

Komponentti alustaa tietokantayhteydet. Komponentti toteutetaan tyhjinä metodikutsuina niiden luokkien ja metodien osalta, jotka kuuluvat osaksi kehysrakennetta. Komponenttia ei toteuteta muilta osin.

2.3.4.1 DatabaseInitializer

DatabaseApplicationInitialize-luokassa alustetaan tietokantayhteydet, ja muut sovelluksen vaatimat alustustoimenpiteet.

`DatabaseApplicationInitializer()`

Konstruktori luokalle.

`void initialize()`

Metodia ei toteuteta. Metodi alustaisi ja mahdollisesti avaisi yhteyden tietokantaan.

2.3.4.2 DatabaseDeInitializer

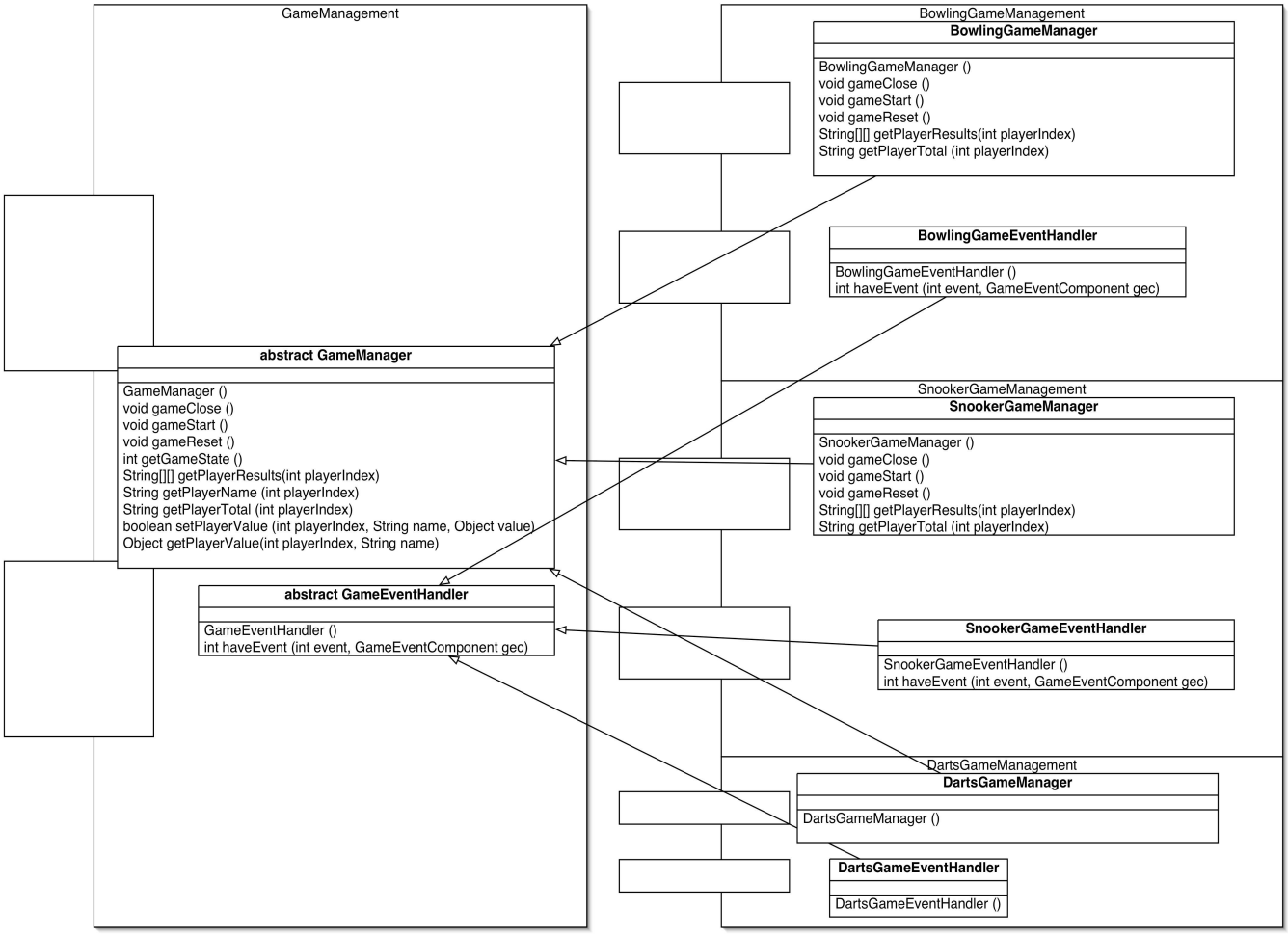
DatabaseApplicationDeinitializer-luokassa suoritetaan sovelluksen sulkemiseen liittyvät toiminnot, kuten mahdolliset tietokantayhteyksien sulkemiset.

`DatabaseDeIinitializer()`
Konstruktori luokalle.

`void deInitialize()`
Metodia ei toteuteta. Metodi sulkisi mahdolliset auki olevat yhteydet tietokantaan.

2.4 GameManagement-komponentti

GameManagement-komponentti toimii välittäjänä (proxy) GUI:n ja muiden komponenttien välillä. GUI:ssa tapahtuvat peliin liittyvät käyttäjän tapahtumat ohjataan kaikki GameEventHandler-luokalle. GUI:n itsensä tekemät kyselyt pelitilanteista kulkevat GameManagement-luokan kautta.



Kuva 2.9: GameManagement-komponentti

2.4.1 GameManager

GameManager-luokka toimii välittäjänä GUI-komponentilta tuleville kutsuille. Kutsut koskevat yleensä pelitilannetta ja pelaajien pistetilanteiden päivitystä. Luokka on abstrakti, ja se tulee toteuttaa jokaisessa sovelluksessa erikseen (vaatimukset 2.3.12 ja 2.3.10). Luokkaa käyttää myös ReportManagement-komponentti tuottaessaan

tuloslistoja.

`GameManager()`

Konstruktori luokalle.

`void gameClose()`

Metodia kutsutaan pelin päättymisen yhteydessä.

`void gameStart()`

Metodia kutsutaan pelin aloituksen yhteydessä, kun halutaan täysi alustus. Täysi alustus tarkoittaa pelaajien nimien uudelleen kysymistä.

`void gameReset()`

Metodi mahdollistaa pelin uudelleen käynnistymisen ilman, että pelaajien nimiä kysytään uudestaan. Pelaajien pisteet ja pelitilanne(GameState) nollataan.

`int getGameState()`

Palauttaa pelin tilannetta kuvaavan arvon.

`String[][] getPlayerResults(int playerIndex)`

Palauttaa pelaajakohtaisen matriisin, joka pitää sisällään pelaajan toistaiseksi kirjatut pisteet ja tarvittavat yhteispisteet, jotta GUI voi päivittää tiedot käyttöliittymään. Matriisin koko on pelikohtainen. Jos pelaajaa ei ole olemassa, metodi heittää IndexOutOfBoundsException–poikkeuksen.

`String getPlayerName(int playerIndex)`

Palauttaa pelaajan nimen. Jos pelaajaa ei ole olemassa, metodi heittää IndexOutOfBoundsException –poikkeuksen.

`String getPlayerTotal(int playerIndex)`

Palauttaa pelaajan kokonaisyhteispisteet. Jos pelaajaa ei ole olemassa, metodi heittää IndexOutOfBoundsException –poikkeuksen.

`boolean setPlayerValue(int playerIndex, String name,`

`Object value)`

Metodin avulla voidaan antaa pelaajille uusia attribuutteja ja attribuuttien arvoja. Metodi tarkistaa kaikki syötetyt attribuutit ja niiden arvot ennen tallentamista Player-luokkaan. Tarkistus tapahtuu kutsumalla Logic-komponentista validaattoria. Palauttaa true, jos lisäys tehtiin. Jos pelaajaa ei ole olemassa, metodi heittää IndexOutOfBoundsException –poikkeuksen.

`Object getPlayerValue(int playerIndex, String name)`

Metodi palauttaa pelaajalle annetun attribuutin arvon. Palauttaa null, jos kysyttyä attribuuttia ei ole pelaajalla. Jos pelaajaa ei ole olemassa, metodi heittää IndexOutOfBoundsException –poikkeuksen.

2.4.2 GameEventHandler

Luokka GameEventHandler ottaa vastaan GUI:ssa tapahtuvat käyttäjän aiheuttamat tapahtumat (event), tulkitsee tapahtuman ja suorittaa tapahtumaan liittyvät toimenpiteet. Luokka on abstrakti, ja jokaisen sovelluksen tulee toteuttaa tämä luokka. Sovelluksilla on yhteisiä ja omia tapahtumia (event), joita välitetään GameEventHandlerille haveEvent-metodin avulla. Kaikille sovelluksille yhteiset tapahtumat löytyvät tämän dokumentin lopusta (Liite 2). GameEventHandlerin ja GUI:n välinen kommunikointi on sovellusten olennaisin osa, kaikki käyttäjälähtöiset tapahtumat ja niiden käsittely kulkee tätä kautta.

`GameEventHandler()`

Konstruktori luokalle.

`int haveEvent(int event, GameEventComponent gec)`

Metodi on pelitapahtumien pääkäsittelijä. Täällä määritellään, mitä toimenpiteitä mikin tapahtuma aiheuttaa. Metodi saa parametreinaan luvun (int), joka yksilöi tapahtumatyypin sekä viitteen siihen komponenttiin, jossa tapahtuma tapahtui (GameEventComponent). Paluuarvonaan metodi palauttaa pelin uuden tilan, tai mikäli aiheutui virhe, metodi palauttaa saamansa event-arvon negatiivisena.

2.4.3 DartsGameManagement-komponentti

Erikoistaa GameManagement-komponentin mökkitikkasovellusta varten.

2.4.3.1 DartsGameManager

Erikoistaa luokan GameManager.

`DartsGameManager()`
Konstruktori luokalle.

2.4.3.2 DartsGameEventHandler

Erikoistaa luokan GameEventHandler.

`DartsGameEventHandler()`
Konstruktori luokalle.

2.4.4. SnookerGameManagement-komponentti

Erikoistaa GameManagement-komponentin Snooker-sovellusta varten.

2.4.4.1 SnookerGameManager

Erikoistaa luokan GameManager.

`SnookerGameManager()`
Konstruktori luokalle.

`void gameStart()`
Metodia kutsutaan pelin aloituksen yhteydessä, kun halutaan täysi alustus. Täysi alustus tarkoittaa pelaajien nimien uudelleen kysymistä.

`void gameReset()`
Metodi mahdollistaa pelin uudelleen käynnistymisen ilman, että pelaajien nimiä

kysytään uudestaan. Pelaajien pisteet ja pelitilanne(GameState) nollataan.

`int getGameState()`

Palauttaa pelin tilannetta kuvaavan arvon.

`String[][] getPlayerResults(int playerIndex)`

Palauttaa pelaajakohtaisen matriisin, joka pitää sisällään pelaajan toistaiseksi kirjatut pisteet ja tarvittavat yhteispisteet, jotta GUI voi päivittää tiedot käyttöliittymään. Matriisin koko on pelikohtainen. Jos pelaajaa ei ole olemassa, metodi heittää IndexOutOfBoundsException–poikkeuksen.

`String getPlayerTotal(int playerIndex)`

Palauttaa pelaajan kokonaisyhteispisteet. Jos pelaajaa ei ole olemassa, metodi heittää IndexOutOfBoundsException–poikkeuksen.

2.4.4.2 SnookerGameEventHandler

Erikoistaa luokan GameEventHandler.

`SnookerGameEventHandler()`

Konstruktori luokalle.

`haveEvent(int event, GameEventComponent gec)`

Metodi on pelitapahtumien pääkäsittelijä. Täällä määritellään, mitä toimenpiteitä mikin tapahtuma aiheuttaa. Metodi saa parametreinaan luvun (int), joka yksilöi tapahtumatyypin sekä viitteen siihen komponenttiin, jossa tapahtuma tapahtui (GameEventComponent).

2.4.5 BowlingGameManagement-komponentti

Erikoistaa GameManagement-komponentin Keilailusovellusta varten.

2.4.5.1 BowlingGameManager

Erikoistaa luokan GameManager.

`BowlingGameManager()`

Konstruktori luokalle.

`void gameClose()`

Metodia kutsutaan pelin päättymisen yhteydessä.

`void gameStart()`

Metodia kutsutaan pelin aloituksen yhteydessä, kun halutaan täysi alustus. Täysi alustus tarkoittaa pelaajien nimien uudelleen kysymistä.

`void gameReset()`

Metodi mahdollistaa pelin uudelleen käynnistymisen ilman, että pelaajien nimiä kysytään uudestaan. Pelaajien pisteet ja pelitilanne(GameState) nollataan.

`String[][] getPlayerResults(int playerIndex)`

Palauttaa pelaajakohtaisen matriisin, joka pitää sisällään pelaajan toistaiseksi kirjatut pisteet ja tarvittavat yhteispisteet, jotta GUI voi päivittää tiedot käyttöliittymään. Matriisin koko on pelikohtainen. Jos pelaajaa ei ole olemassa, metodi heittää IndexOutOfBoundsException–poikkeuksen.

`String getPlayerTotal(int playerIndex)`

Palauttaa pelaajan kokonaisyhteispisteet. Jos pelaajaa ei ole olemassa, metodi heittää IndexOutOfBoundsException–poikkeuksen.

2.4.5.2 BowlingGameEventHandler

Erikoistaa luokan GameEventHandler.

`BowlingGameEventHandler()`

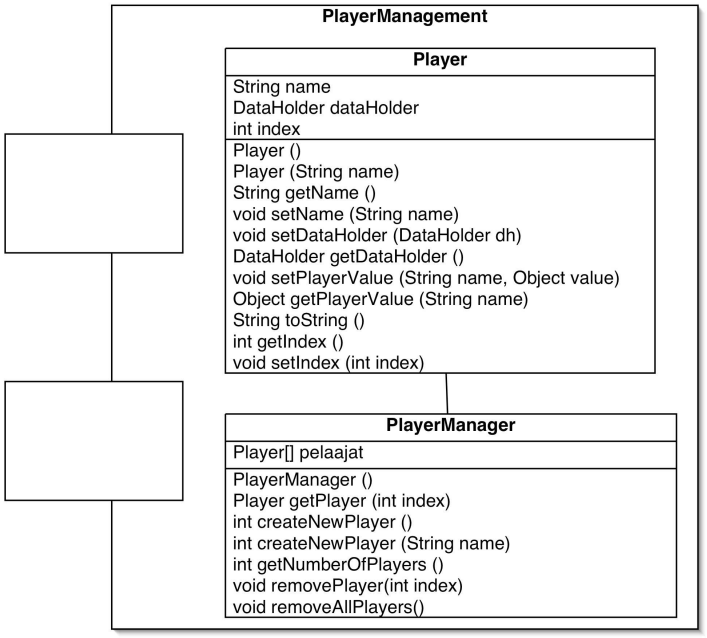
Konstruktori luokalle.

`haveEvent(int event, GameEventComponent gec)`

Metodi on pelitapahtumien pääkäsittelijä. Täällä määritellään, mitä toimenpiteitä mikin tapahtuma aiheuttaa. Metodi saa parametreinaan luvun (int), joka yksilöi tapahtumatyypin sekä viitteen siihen komponenttiin, jossa tapahtuma tapahtui (GameEventComponent).

2.5 PlayerManagement-komponentti

Komponentti huolehtii pelaajahallinnasta, eikä siitä ole erikoistettavia toteutuksia. Player-luokka tallentaa pelaajakohtaisia tietoja. PlayerManager-luokka pitää viittauksia kaikkiin sovellusten pelin aikaisiin pelaajiin, kaikki pelaajat luodaan tämän luokan kautta.



Kuva 2.10: Komponentti PlayerManagement

2.5.1 Player

Player-luokka pitää sisällään tiedon pelaajakohtaisista muuttujista, kuten nimi ja sukupuoli. Player-luokalla on myös viite omaan ResultManager komponentin DataHolder-luokkaan, jonka avulla pelaajan pisteistä pidetään kirjaa.

`Player()`
Konstruktori luokan ilmentymän luomista varten.

`Player(String name)`
Konstruktori luokan luomista varten. Nimeää samalla pelaajan.

`String getName()`
Palauttaa pelaajan nimen

`void setName(String name)`
Asettaa pelaajalle uuden nimen.

`void setDataHolder(Dataholder dh)`
Asettaa pelaajalle DataHolder-olion, jota käytetään pelaajan pisteiden tallentamiseen.

`Dataholder getDataHolder()`
Palauttaa viitteen pelaajakohtaiseen pistekanta-alkioon.

`void setPlayerValue(String name, Object value)`
Asettaa pelaajalle uuden attribuutin. Jos name on null, metodi heittää IllegalArgumentException-poikkeuksen.

`Object getPlayerValue(String name)`
Palauttaa parametrina annetun attribuutin arvon. Arvo on null, jos attribuuttia ei ole pelaajalla. Jos name on null, heittää IllegalArgumentException poikkeuksen.

`String toString()`

Kuormittaa oletusarvoisen toString()-metodin ja palauttaa pelaajan nimen.

`int getIndex()`

Palauttaa pelaajan index-arvon. Toisilla pelaajilla ei saa olla samaa index-arvoa. Palauttaa null, jos arvoa ei ole asetettu.

`void setIndex(int index)`

Asettaa pelaajalle index-arvon. Toisilla pelaajilla ei saa olla samaa index-arvoa. Tarkistus on asettajan vastuulla.

2.5.2 PlayerManager

Luokka pitää kirjaa kaikista pelin pelaajista. Kaikki uudet pelaajat tulee luoda tämän luokan metodien kautta, samoin poiston tulee tapahtua tämän luokan kautta. Luokka kutsuu Player-luokan ilmentymää luodessaan itse ComponentReference-luokkaa. Tämä mahdollistaa Player-luokan erikoistamisen myöhemmin, kun joku sovellus sitä tarvitsee.

`PlayerManager()`
Konstruktori luokalle.

`Player getPlayer(int index)`
Palauttaa viitteen index-numeroa vastaavaan Player-luokan olioon. Jos pelaajaa ei ole olemassa, metodi heittää IndexOutOfBoundsException –poikkeuksen.

`int createNewPlayer()`
Luo uuden pelaajan. Kutsuu itse ComponentReference-luokkaa, joten tätä metodia voi kutsua suoraan. Uutta pelaajaa luodessa, tämän metodin tulisi samalla asettaa pelaajalle DataHolder-viite ja index-arvo. Metodi palauttaa arvonaan uuden pelaajan index-arvon.

`int createNewPlayer(String name)`
Luo uuden pelaajan pelaaja nimellä. Metodi kutsuu ComponentReference-luokkaa, joten tätä metodia voi kutsua suoraan. Uutta pelaajaa luodessa tämän metodin tulisi samalla asettaa pelaajalle DataHolder-viite ja index-arvo. Metodi palauttaa arvonaan uuden pelaajan index-arvon.

`int getNumberOfPlayers()`
Palauttaa arvonaan pelaajamäärän.

`void removePlayer(int index)`

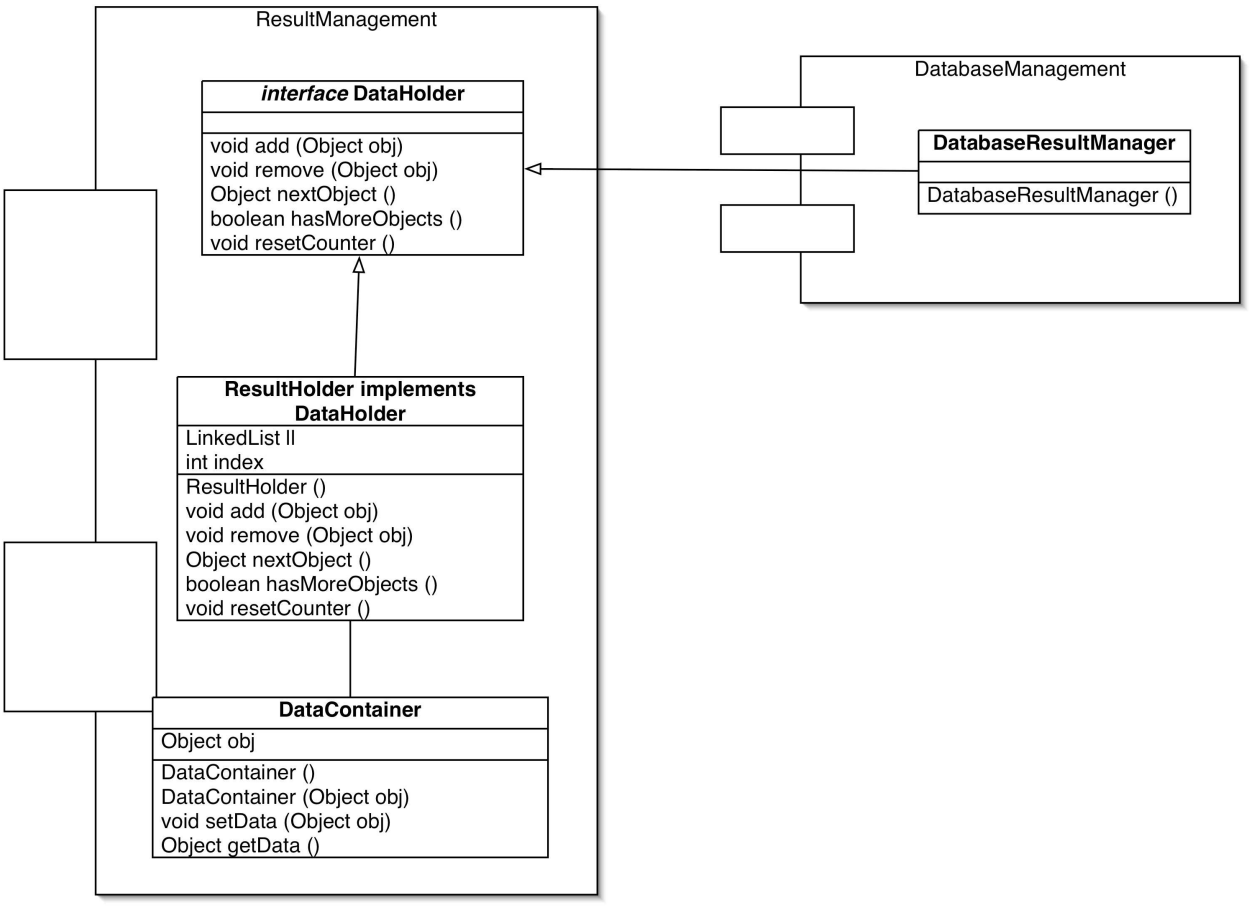
Poistaa index-arvoa vastaavan pelaajan. Muiden pelaajien index-arvo ei muutu. Jos pelaajaa ei ole olemassa, metodi heittää IndexOutOfBoundsException–poikkeuksen. Jos index on null, metodi heittää IllegalArgumentException-poikkeuksen.

`void removeAllPlayers()`
Poistaa kaikki pelaajat.

2.6 ResultManagement-komponentti

Komponentin avulla määritellään tietorakennetoteutus, jota käytetään pisteiden hallinnointiin sovelluksen ajon aikana. Komponenttia ei erikoisteta projektin aikana toteutettavia sovelluksia varten, vaan perustoteutuksena on linkitettylista, jota kaikki sovellukset käyttävät. Komponentissa on kuitenkin erikoistettavia kohtia myöhempää tarvetta varten. Nämä kohdat ovat DataHolder ja DataContainer.

DataHolder-rajapintaluokka määrittelee mitkä metodit tietorakenteen (tässä tapauksessa linkitettylista) on toteutettava. ResultHolder toteuttaa DataHolder-rajapintaluokan metodit linkitetylle listalle. DataContainer-luokka on yksinkertainen papu (Bean), joka toimii alkiona DataHolder-rajapinnan mukaisissa tietorakennetoteutuksissa.



Kuva 2.11: ResultManagement-komponentti

2.6.1 DataHolder

Rajapinta (Interface) DataHolder määrittää ne perusmetodit, joita tietorakenne-ratkaisun tulisi toteuttaa, jotta ratkaisu olisi yhtenevä ohjelmistokehyksen kanssa. Rajapinta on myös erikoistamiskohta, mutta ainoastaan perustoteutusta käytetään nyt esillä olevien toteutusten yhteydessä. Tämä luokka ei ole säieturvallinen.

`void add(Object obj)`

Metodin avulla lisätään uusi DataContainder tietorakenteesta.

`void remove(Object obj)`

Metodin avulla poistetaan DataContainer tietorakenteesta. Jos parametri on null, tulisi heittää IllegalArgumentException-poikkeus.

`Object nextContainer()`

Palauttaa viitteen seuraavaan talletettuun alkioin. Takaisin alkuun pääsee kutsumalla resetCounter()-metodia. Toteutus on yksinkertainen iteraattori sillä erotuksella, että iteraattoreita voi olla kerrallaan vain yksi. Metodi heittää NoSuchElementException-poikkeuksen, jos luokkia ei enää ole.

`boolean hasMoreContainers()`

Paluuarvo kertoo, onko tietorakenteessa vielä jäljellä DataContainer-alkioita, joita ei ole läpikäyty. Tällä voidaan välttää turhia nextContainer-kutsuja. Metodi palauttaa arvon true, jos on DataContainer-alkioita jäljellä, ja false, jos ei ole.

`void resetCounter()`

Tällä metodilla voidaan palauttaa iteraattori takaisin alkiojonon alkuun.

2.6.2 ResultHolder

ResultHolder on ohjelmistoperheessä nyt käytettäviä sovelluksia varten tehty perustoteutus pisteiden tallennukseen käytettävästä tietorakenteesta. Luokka toteuttaa rajapinnan DataHolder. Toteutettava tietorakenne on linkitetty lista. Samoin luokka toteuttaa rajapinnan DataHolder mukaiset metodit.

2.6.3 DataContainer

DataContainer-luokka on yksinkertainen papu (Bean), joka toimii alkiona DataHolder-rajapinnan mukaisissa tietorakennetoteutuksissa. DataContainer-luokka on tarkoitettu varastoimaan pelaajien pelin aikana keräämiä pisteitä, ei kuitenkaan yhteen laskettuja pisteitä.

`DataContainer()`
Konstruktori luokalle.

`DataContainer(object obj)`
Konstruktori saa parametrinaan talletettavan objektin.

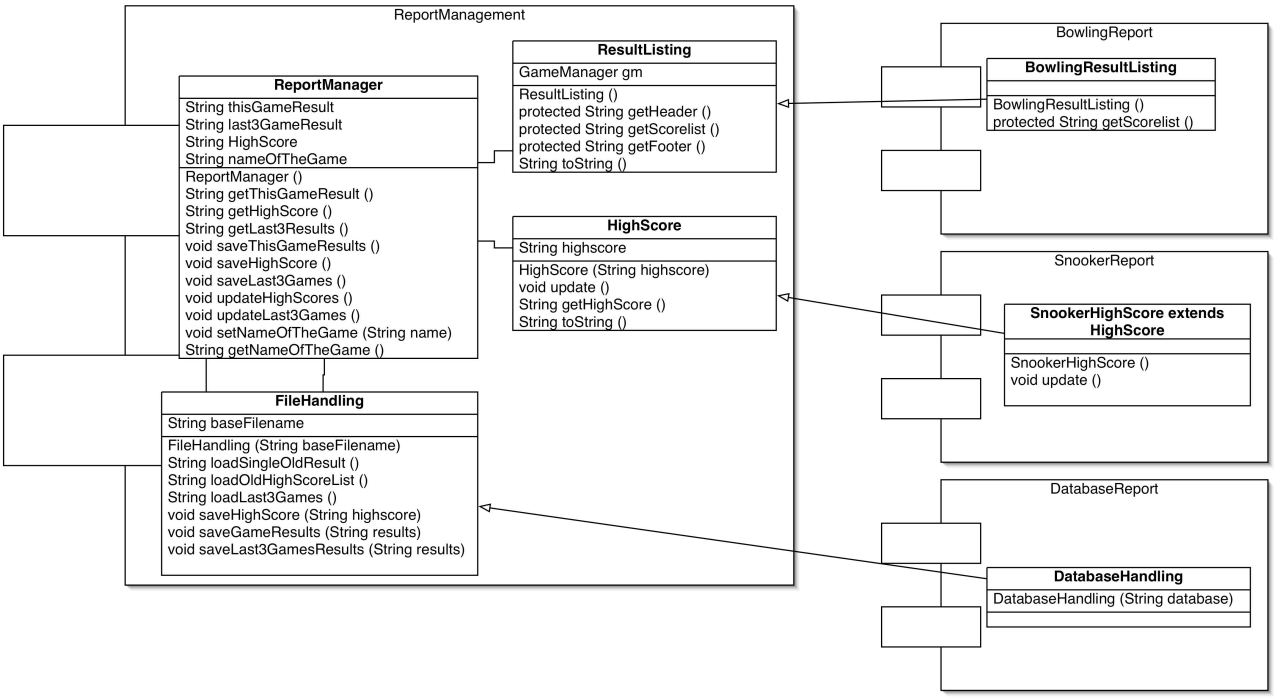
`void setData(object obj)`
Annetaan alkiolle objekti talletettavaksi. Alkio voi pitää sisällään vain yhtä objektia, joten vanha poistuu, jos sellainen oli.

`Object getData()`
Palauttaa viitteen alkiossa olevaan objektiin.

2.7 ReportManagement-komponentti

ReportManagement-komponentti huolehtii pelitulosten tallentamisesta ja lukemisesta. Komponentin avulla voidaan myös muotoilla tuloslista, ja valita käytettävien tuloslistojen muodot. Erilaisia muotoja ovat viimeisin peli, kolme viimeisintä peliä, tai parhaat tulokset (highscore). Komponentissa on useita eri erikoistamiskohtia.

ReportManager-luokka toimii välittäjänä (proxy) komponentin eri luokkien palveluille. FileHandling-luokka määrittelee toimenpiteet, joilla sovelluksen pisteet voidaan tallentaa ja ladata. Tiedostorakenteet on kuvattu liitteessä 3. ResultListing–luokka tuottaa varsinaisen valmiiksi muotoillun tuloslistan sille annetuista pisteistä. Luokka on erikoistettavissa perinnän avulla. Luokan avulla voidaan tuottaa paras tulos -tyylisiä tuloslistoja ja määritellä peleistä mitkä ovat ne ominaisuudet, joilla paras tulos määritellään.



Kuva 2.12: ReportManagement-komponentti

2.7.1 ReportManager

Luokka toimii välittäjänä (Proxy) komponentin tarjoamille toiminnoille.

`ReportManager()`
Konstruktori luokalle.

`String getThisGameResult()`
Palauttaa tuloslistan viimeksi pelatusta pelistä.

`String getHighScore()`
Palauttaa parhaat pisteet -tyylisen tuloslistan.

`String getLast3Results()`
Palauttaa tuloslistan, jossa on eritelty tulokset kolmesta viime pelistä.

`void saveThisGameResults()`
Tallentaa viimeksi pelatun pelilistan.

`void saveHighscore()`
Tallentaa parhaat pisteet.

`void saveLast3Games()`
Tallentaa kolme viimeksi pelattua pistelistaa. Katso (Liite 3).

`void updateHighScores()`
Päivittää viimeksi pelatusta pelistä parhaat pisteet yhdistettyyn parhaat pisteet -listaan.

`void updateLast3Games()`
Päivittää kolme tuloslistaa lisäämällä alkuun viimeksi pelatun pelin tuloslistan ja poistamalla vanhimman tuloslistan, mikäli tuloslistoja on jo kolme ennen päivitystä. Jokainen tuloslista on eroteltu `<separator>`-merkinnällä.

`void setNameOfTheGame(String name)`

Metodilla kerrotaan Report-komponentille sovelluksen nimi.

`String getNameOfTheGame()`

Metodi on tarkoitettu lähinnä Report-komponentin sisäiseen käyttöön.

2.7.2 FileHandling

Luokka määrittelee toimenpiteet, joilla sovelluksen pisteet voidaan tallentaa ja ladata. Luokka on erikoistettavissa perimällä (extends). Tiedostorakenteet on kuvattu liitteessä 3.

`FileHandling(String baseFilename)`

Metodi luo FileHandling-olion. Parametrina annetaan sovelluksen tiedostonimissä käyttämä tunnus, jonka avulla tallennustiedostot erotellaan sovellusten kesken. Tunnus on esimerkiksi ”darts”. Jos baseFilename on null, heittää IllegalArgumentException-poikkeuksen.

`String loadSingleOldResult()`

Palauttaa arvonaan tiedostosta ladatun viimeksi pelatun pelin tuloslistan.

`String loadOldHighScoreList()`

Palauttaa arvonaan tiedostosta ladatun parhaat tulokset -listan.

`String LoadLast3Games()`

Palauttaa arvonaan tiedostosta ladatun listan, jossa on kolmen viimeisen pelin tulokset.

`void saveHighSCore(String HighScore)`

Tallentaa parhaat pisteet.

`void saveGameResults(String results)`

Tallentaa viimeksi pelatun pelin tuloslistan.

`void saveLast3GamesResults(String results)`
Tallentaa kolmen viimeksi pelatun pelin tuloslistat tai niin monta kuin on saatavilla, mikäli tuloslistoja on alle kolme.

2.7.3 ResultListing

ResultListing–luokka tuottaa varsinaisen valmiiksi muotoillun tuloslistan sille annetuista pisteistä. Luokka on erikoistettavissa perinnän avulla.

`ResultListing()`
Konstuktori luokalle.

`protected String getHeader()`
Luo ja palauttaa tuloslistan otsikon.

`protected String getScorelist()`
Luo ja palauttaa varsinaisen tuloslistan.

`protected String getFooter()`
Luo ja palauttaa jälkitekstin.

`String toString()`
Palauttaa Stringin, jossa on yhdistettynä header, scorelist ja footer.

2.7.4 HighScore

Luokan avulla voidaan tuottaa paras tulos -tyylisiä tuloslistoja ja määritellä peleistä mitkä ovat ne ominaisuudet, joilla paras tulos määritellään. Perustoteutus on suurin yhteispistemäärä.

`Highscore(String Highscore)`
Konstruktori saa luontivaiheessa parametrinaan vanhat parhaat pisteet, jotka on luettu FileHandlinging kautta.

`void update()`
Päivittää parhaat pisteet viimeksi loppuneen (tai nyt käynnissä olevan) pelin pisteisiin.

`String getHighscore()`
Palauttaa parhaat pisteet tiedostoon talletettavassa muodossa.

`String toString()`
Palauttaa parhaat pisteet GUI:lle muokatussa muodossa. Täällä siis määritellään tuloslistan ulkoasu.

2.7.5 BowlingReport-komponentti

Komponentti erikoistaa Result-komponentin yhden luokan osalta. BowlingResultListing perii ResultListing-luokan.

BowlingResultListing

Luokka perii ResultListing luokan ja täten erikoistaa perustoteutusta. Luokan erikoistamisella halutaan luoda keilailusovellusta varten oma tuloslistan ulkoasu.

`BowlingResultListing()`
Konstuktori luokalle.

```
protected String getScorelist();
```

Luo ja palauttaa varsinaisen tuloslistan.

2.7.6 SnookerReport-komponentti

Komponentti erikoistaa Result-komponentin yhden luokan osalta. SnookerHighscore-luokka perii HighScore-Luokan.

SnookerHighScore

Luokka perii Highscore-luokan ja korvaa metodin update(). Erikoistamisella saadaan aikaiseksi toiminto, jolla paras tulos mittaa yhteistuloksen sijasta parasta yhden vuoron aikana kerättyä pistesaalista (paras break).

`Highscore(String Highscore)`
Konstruktori saa luontivaiheessa parametrinaan vanhat parhaat pisteet, jotka on luettu FileHandlingin kautta.

`void update()`
Päivittää parhaat pisteet viimeksi loppuneen (tai nyt käynnissä olevan) pelin pisteisiin.

2.7.7 DatabaseReport-komponentti

Komponentti erikoistaa Result-komponentin yhden luokan osalta. DatabaseHandling erikoistaa FileHandling-luokan. Komponentista tehdään vain tyhjä toteutus.

DatabaseHandling

Luokka erikoistaa FileHandling-luokan perimällä sen. Luokasta tehdään vain tyhjä toteutus.

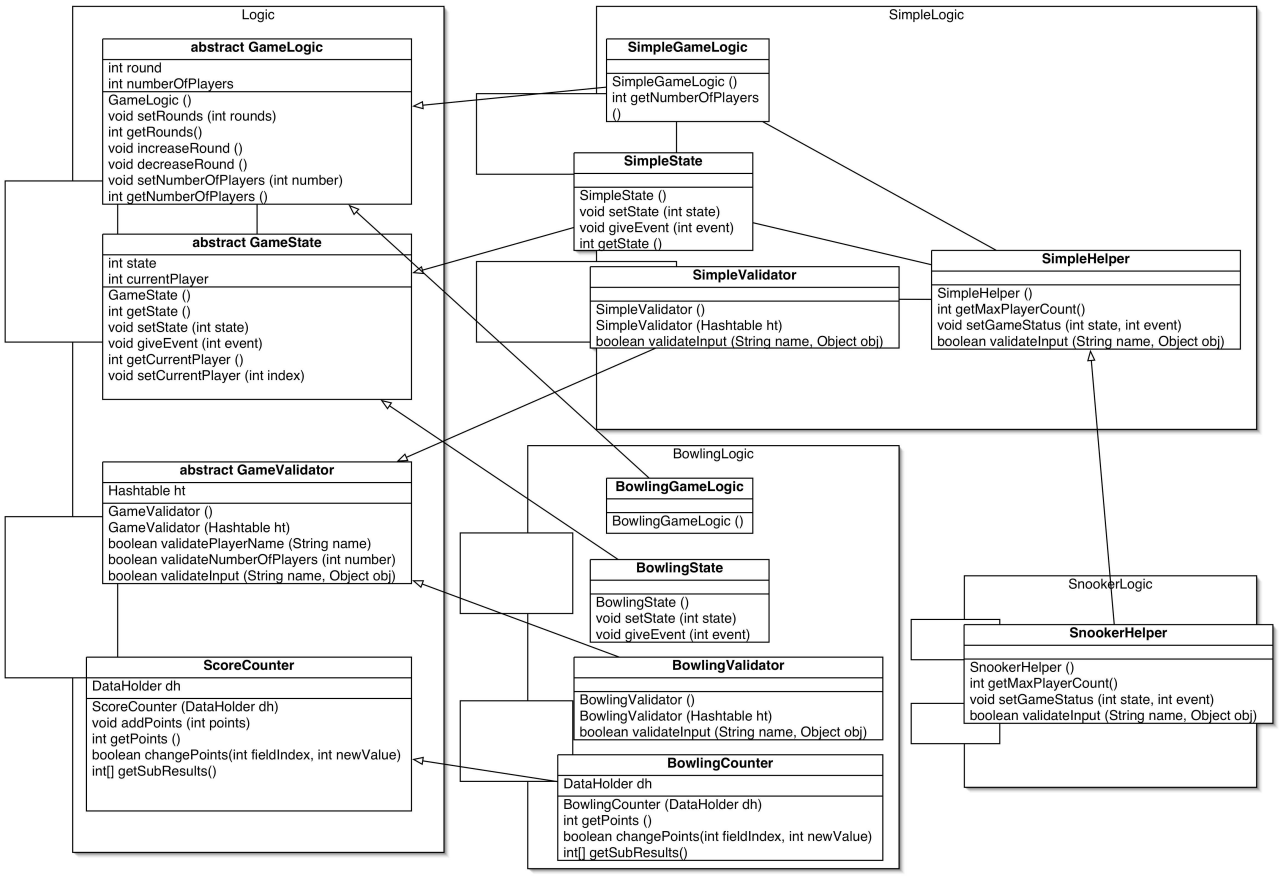
`DatabaseHandling(String baseFilename)`

Metodi luo DatabaseHandling-olion. Parametrina annetaan sovelluksen tunnisteena käyttämä tunnus, jonka avulla tulokset erotellaan sovellusten kesken. Tunnus on esimerkiksi ”darts”.

2.8 Logic-komponentti

Logic-komponentti huolehtii pelin sääntöjen noudattamisen valvonnasta, pelitilanteen seuraamisesta, pisteiden yhteenlaskutavasta, sekä käyttäjän antamien syötteiden tarkistuksesta. Komponentti ja siitä erikoistetut komponentit muodostavat syvimmillään kolmikerroksisen toteutuksen (vaatimus 2.3.9). Kolmikerroksinen toteutus toteutus näkyy myös kuvassa selkeästi. Kerroksia ei ylitetä, koska SnookerLogic-komponentin erikoistettavat osat on kaikki erikoistettu myös SimpleLogic-komponentissa (vaatimus 2.3.7).

GameLogic-luokka pitää kirjaa tämänhetkisestä kierroksesta ja pelaajamäärästä. GameState-luokka pitää kirjaa tämänhetkisestä pelitilanteesta. GameValidator-luokka tarkistaa syötteitä sekä varastoi tietoonsa eri syötteisiin käyvät raja-arvot. ScoreCounter-luokka määrittelee, kuinka pelaajan syöttämät syötteet suhteutetaan toisiinsa loppu- ja välitulosten saamiseksi. Jokaista pelaajaa kohden luodaan oma ScoreCounter. SimpleLogicErikoistaa Logic-komponentin luokat GameLogic, GameState ja GameValidator.



Kuva 2.13: Logic-komponentti

2.8.1 GameLogic

Luokka pitää kirjaa tämänhetkisestä kierroksesta ja pelaajamäärästä. Jos pelissä on kierroksia, kierrosmäärän muuttamisesta huolehtii lähinnä GameState-luokka. Kierrosmäärä tulisi nollata, kun pelitilanne palautuu alkutilaa vastaavaksi.

`GameLogic()`
Konstruktori luokalle.

`void setRounds(int rounds)`
Asettaa kierrosmäärän annettuun arvoon. Tätä voidaan käyttää kierrosluvun nollaamiseksi.

`int getRounds()`
Palauttaa kierrosmäärän.

`void setNumberOfPlayers(int number)`
Asettaa peliin otettujen pelaajien määrän.

`int getNumberOfPlayers()`
Palauttaa pelaajien lukumäärän. Kutsuu PlayerManageria.

`void increaseRound()`
Nostaa kierroslukua yhdellä

`boolean decreaseRound()`
Laskee kierroslukua yhdellä. Palauttaa false, jos kierrosluku oli jo 0 eli alin mahdollinen. Muuten palauttaa true.

2.8.2 GameState

Luokka pitää kirjaa tämänhetkisestä pelitilanteesta. Kaikki pelitilanteeseen vaikuttavat tapahtumat tulisi ohjata myös tämän luokan tietoisuuteen kutsumalla giveEvent()-metodia.

`GameState()`
Konstruktori luokalle.

`void setState(int state)`
Asettaa uuden pelitilanteen tilan. Tämä tulisi asettaa manuaalisesti lähinnä silloin, kun peli aloitetaan alusta tai peli loppuu.

`int getState()`
Palauttaa nykyisen pelitilanteen tilan.

`void setCurrentPlayer(int playerIndex)`
Asettaa parametrinaan saaman pelaajan index-arvon aktiiviseksi pelaajaksi. Jos pelaajaa ei ole olemassa, heittää IndexOutOfBoundsException–poikkeuksen.

`int getCurrentPlayer()`
Palauttaa aktiivisena olevan pelaajan index-arvon

`void giveEvent(int event)`
Antaa parametrinaan käyttäjän syöttämän tapahtuman tunnuksen. Laskee uuden gameState-arvon.

2.8.3 GameValidator

Luokka tarkistaa syötteitä sekä varastoi tietoonsa eri syötteisiin käyvät raja-arvot.

`GameValidator()`
Konstuktori luokalle.

`GameValidator(Hashtable ht)`

Konstruktori, joka saa hajautustaulun (hashtable) parametrinaan. Hajautustaulua voidaan käyttää sallittujen syötteiden arvojen tarkistamiseen.

`boolean validatePlayerName(String name)`

Tarkistaa pelaajan nimen oikeellisuuden. Palauttaa false, jos nimi on virheellinen, ja true, jos nimi on oikea.

`boolean validateNumberOfPlayers(int number)`

Tarkistaa, onko käyttäjän syöttämä pelaajamäärä sallittu.

`boolean validateInput(String name, Object obj)`

Tarkistaa, onko käytettävän attribuutin (name) arvo (obj) sallittu. Palauttaa true, mikäli arvo on sallittu, ja false, mikäli ei ole sallittu. Jos name on null, heittää IllegalArgumentException poikkeuksen.

2.8.4 ScoreCounter

Luokka määrittelee, kuinka pelaajan syöttämät syötteet suhteutetaan toisiinsa loppu- ja välitulosten saamiseksi. Jokaista pelaajaa kohden luodaan oma ScoreCounter.

`ScoreCounter(DataHolder dh)`

Konstuktori saa pelaajakohtaisen DataHolder ilmentymän parametrinaan.

`void addPoints(int points)`

Lisää pisteitä pelaajalle.

`int getPoints()`

Palauttaa pelaajan kokonaispisteet.

`int[] getSubResults()`

Palauttaa pelaajan välitulokset. Perustoteutuksessa palautetaan kaikki muut pelaajan pisteet paitsi yhteistulos.

```
boolean changePoints(int fieldIndex, int newValue)
```

Metodilla voidaan muokata jo kertaalleen syötettyjä pisteitä. Metodi saa arvonaan syötekentän numeron, jota muokataan (fieldIndex), sekä kenttään sijoitetut korjatut pisteet (newValue). Jos fieldIndex on virheellinen, metodi heittää IndexOutOfBoundsException–poikkeuksen.

2.8.5 SimpleLogic-komponentti

Erikoistaa Logic-komponentin luokat GameLogic, GameState ja GameValidator.

2.8.5.1 SimpleLogic

Erikoistaa luokan GameLogic.

```
SimpleLogic()
```

Konstruktori luokalle.

2.8.5.2 SimpleState

Erikoistaa luokan GameState.

```
SimpleState()
```

Konstruktori luokalle.

2.8.5.3 SimpleValidator

Erikoistaa luokan GameValidator.

```
SimpleValidator()
```

Konstuktori luokalle.

```
SimpleValidator(Hashtable ht)
```
Konstuktori saa parametrinaan hajautustaulun (hashtable).

```
boolean validateInput(String name, Object obj)
```
Tarkistaa, onko käytettävän attribuutin (name) arvo (obj) sallittu. Palauttaa true, mikäli arvo on sallittu, ja false, mikäli ei ole sallittu.

2.8.5.4 SimpleHelper

Luokka toimii apuluokkana muille komponentin luokille.

```
SimpleHelper()
```
Konstuktori luokalle.

```
int getMaxPlayerCount()
```
Metodi palauttaa maksimipelaajamäärän pelissä.

```
void setGameStatus(int state, int event)
```
Metodilla asetetaan uusi pelin tila.

```
boolean validateInput(String name, Object obj)
```
Metodi kuormittaa saadun syötteen tarkistajan. Palauttaa true, jos arvo on laillinen, ja false, jos laiton. Jos name on null, heittää IllegalArgumentException-poikkeuksen.

2.8.6 SnookerLogic-komponentti

Komponentti erikoistaa SimpleLogic-komponentin. Erikoistaa luokan SimpleHelper.

SnookerHelper

Erikoistaa luokan SimpleHelper.

```
SnookerHelper()
```

Konstuktori luokalle.

`int getMaxPlayerCount()`

Metodi palauttaa maksimipelaajamäärän pelissä.

`void setGameStatus(int state, int event)`

Metodilla asetetaan uusi pelin tila.

`boolean validateInput(String name, Object obj)`

Metodi kuormittaa saadun syötteen tarkistajan. Palauttaa true, jos arvo on laillinen, ja false, jos laiton. Jos name on null, heittää IllegalArgumentException-poikkeuksen.

2.8.7 BowlingLogic-komponentti

Erikoistaa komponentin Logic. Erikoistettavat luokat ovat GameState, GameValidator ja ScoreCounter.

2.8.7.1 BowlingState

Erikoistaa luokan GameState.

`BowlingState()`

Konstuktori luokalle.

`int getState()`

Palauttaa nykyisen pelitilanteen tilan.

`void giveEvent(int event)`

Antaa parametrinaan käyttäjän syöttämän tapahtuman tunnuksen. Laskee uuden gameState-arvon.

2.8.7.2 BowlingValidator

Erikoistaa luokan GameValidator.

`BowlingValidator()`

Konstuktori luokalle.

`BowlingValidator(Hashtable ht)`

Konstuktori saa parametrinaan hajautustaulun(hashtable).

`boolean validateInput(String name, Object obj)`

Tarkistaa, onko käytettävän attribuutin (name) arvo (obj) sallittu. Palauttaa true, mikäli arvo on sallittu, ja false, mikäli ei ole sallittu. Jos name on null, heittää IllegalArgumentException-poikkeuksen.

2.8.7.3 BowlingCounter

Erikoistaa luokan ScoreCounter.

`BowlingCounter()`

Konstuktori luokalle.

`void addPoints(int points)`

Lisää pisteitä pelaajalle.

`int getPoints()`

Palauttaa pelaajan kokonaispisteet.

`int getPoints(int fromIndex, int toIndex)`

Palauttaa yhteenlasketut pisteet annetulta väliltä. Palauttaa null, jos välillä ei ole pisteitä. Jos pisteväli on virheellinen, heittää IndexOutOfBoundsException-poikkeuksen.

`int[] getSubResults()`

Palauttaa pelaajan välitulokset. Bowling toteutuksessa metodi palauttaa kaikki pistelomakkeen alemmalla kerroksella olevat tulokset.

```
boolean changePoints(int fieldIndex, int newValue)
```

Metodilla voidaan muokata jo kertaalleen syötettyjä pisteitä. Metodi saa arvojaan syötekentän numeron, jota muokataan (fieldIndex), sekä kenttään sijoitetut korjatut pisteet (newValue). Jos fieldIndex on virheellinen, heittää IndexOutOfBoundsException–poikkeuksen.

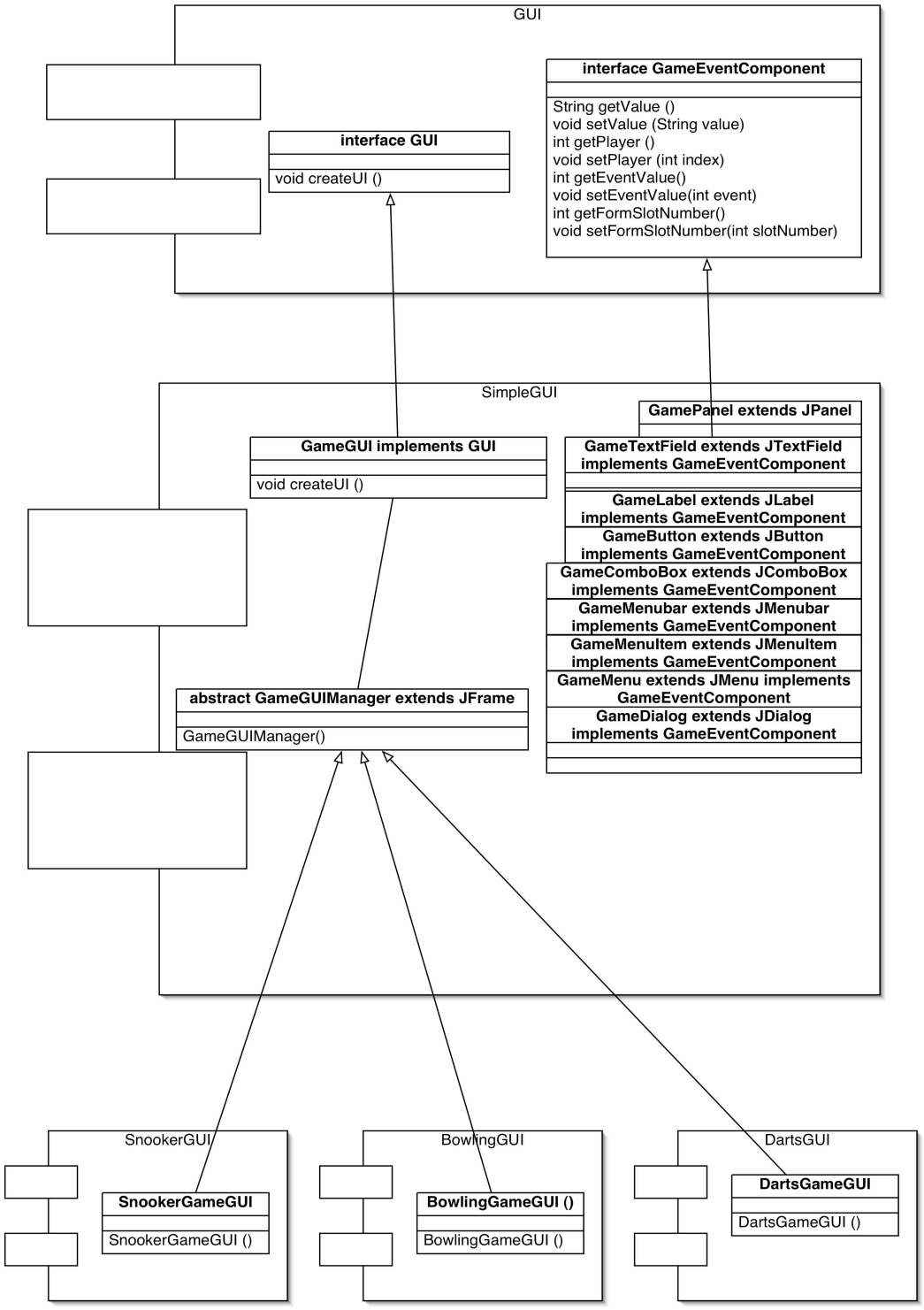
2.9 GUI-komponentti

Komponentti sisältää käyttöliittymän toteutukseen tarvittavan rakenteen.

Gui-komponentti määrittelee kaksi rajapinta-luokkaa GUI ja GameEventComponent.

Kyseiset rajapinnat toteutetaan SimpleGui komponentissa. SimpleGuin GameGui-luokka toteuttaa GUI rajapinnan createUI()-metodin kutsumalla GameGuiManager-luokan createApplicationUI()-metodia. SimpleGui komponentissa myös käyttöliittymäkomponentti kirjasto. Kirjastoa käytetään käyttöliittymän tekstikenttien ym. luomiseen. Jokainen kirjaston käyttöliittymäkomponentti toteuttaa GameEventComponent-rajapinnan.

Sovellusten GameGui-luokat perivät GameGuiManager-luokan ja muodostavat käyttöliittymäkomponentti-kirjaston komponenteilla oman käyttöliittymänsä. Sovelluskohtaiset tapahtumankuuntelijat määritellään myös niiden GameGui-luokissa.



Kuva 2.9: GUI-komponentti

2.9.1 GUI

Rajapinta (interface) GUI määrittelee käyttöliittymäkomponentin perustoteutuksessa ohjelmistokehyksenkannalta pakolliset metodit.

`GUI createUI()`

Luo pelin käyttöliittymän.

2.9.2 GameEventComponent

GameEventComponent-rajapinta määrittelee metodit, jotka peleissä käytettävien käyttöliittymäkomponenttien tulisi toteuttaa.

`String getValue()`

Palauttaa käyttöliittymäkomponentin arvon.

`void setValue()`

Asettaa käyttöliittymäkomponentille arvon.

`int getPlayer()`

Palauttaa pelaajan käyttöliittymäkomponenttiin liitetyn pelaajaindeksin.

`void setPlayer()`

Liittää käyttöliittymäkomponenttiin pelaajaindeksin.

`int getEventValue()`

Palauttaa käyttöliittymäkomponentin tapahtuman indeksin.

`void setEventValue(int event)`

Liittää käyttöliittymäkomponenttiin tapahtuman indeksin.

`int getFormSlotNumber()`

Palauttaa käyttöliittymäkomponentin järjestysnumeron.

`void setFormSlotNumber(int slotNumber)`

Asettaa käyttöliittymäkomponentille järjestysnumeron.

2.9.3 SimpleGui-komponentti

SimpleGui-komponentti tarjoaa graafisen käyttöliittymän tarvitsemia komponentteja. Komponentti toimii myös kirjastona erilaisille sovellusten käyttämille peruskäyttöliittymäkomponenteille. Käyttöliittymäkomponentit toimivat luonteeltaan kirjastomaisesti. Ne perivät Swing-pohjaisen käyttöliittymäkomponentin ja implementoivat GameEventComponent-rajapinnan. GameGui ja GameGUIManager kuuluvat myös SimpleGui-komponenttiin.

2.9.4 GameGUI

Luokka toteuttaa rajapinnan GUI ja alustaa uuden sovelluksen.

`GameGUI()`
Konstruktori luokalle.

`GUI createUI()`

Metodi luo ComponentReferencen avulla GameGUIManager -luokan kutsuen
sen konstruktoria.

2.9.5 GameGUIManager

Luokka laajentaa Swing-luokkaa JFrame ja toimii varsinaisen käyttöliittymän perustana. GameGuiManager on abstrakti luokka, jonka sovellukset erikoistavat.

`GameGUIManager()`
Konstruktori luokalle.

`void alert(String text)`

Käytetään sovelluksen virhetilanteissa virheilmoitusten luontiin.

`boolean confirm(String text)`

Käytetään sovelluksen kyllä/ei-kysymysten esittämiseen.

2.9.6 SimpleGUI:n käyttöliittymäkomponenttikirjasto

Kaikki kirjaston käyttöliittymäkomponentit toteuttavat GameEventComponent rajapinnan määrittelemät metodit ja perivät samalla vastaavan Javan Swing-luokan metodit. Esimerkkinä on esitelty ainoastaan luokka GameTextField.

GameTextField

Laajentaa JTextField-luokkaa ja toteuttaa GameEventComponent-rajapinnan metodit.

`String getValue()`

Palauttaa käyttöliittymäkomponentin arvon.

`void setValue()`

Asettaa käyttöliittymäkomponentille arvon.

`int getPlayer()`

Palauttaa pelaajan käyttöliittymäkomponenttiin liitetyn pelaajaindeksin.

`void setPlayer()`

Liittää käyttöliittymäkomponenttiin pelaajaindeksin.

`int getEventValue()`

Palauttaa käyttöliittymäkomponentin tapahtuman indeksin.

`void setEventValue(int event)`

Liittää käyttöliittymäkomponenttiin tapahtuman indeksin.

`int getFormSlotNumber()`

Palauttaa käyttöliittymäkomponentin järjestysnumeron.

`void setFormSlotNumber(int slotNumber)`

Asettaa käyttöliittymäkomponentille järjestysnumeron.

2.9.7 SnookerGUI-komponentti

Erikoistaa GameGUIManager-komponentin.

SnookerGameGUI

SnookerGameGUI erikoistaa GameGUI-luokan.

`SnookerGameGUI()`

Luo Snookerin käyttöliittymän eli kokoaa käyttöliittymäkohtaiset käyttöliittymäkomponentit.

2.9.8 DartsGui-komponentti

Erikoistaa GameGUIManager-komponentin.

DartsGameGUI

DartsGameGUI erikoistaa GameGUI-luokan.

`DartsGameGUI()`

Luo mökkitikan käyttöliittymän eli kokoaa käyttöliittymäkohtaiset käyttöliittymäkomponentit.

2.9.9 Bowling-komponentti

Erikoistaa GameGUIManager-komponentin.

BowlingGameGui

BowlingGameGui erikoistaa GameGUI-luokan.

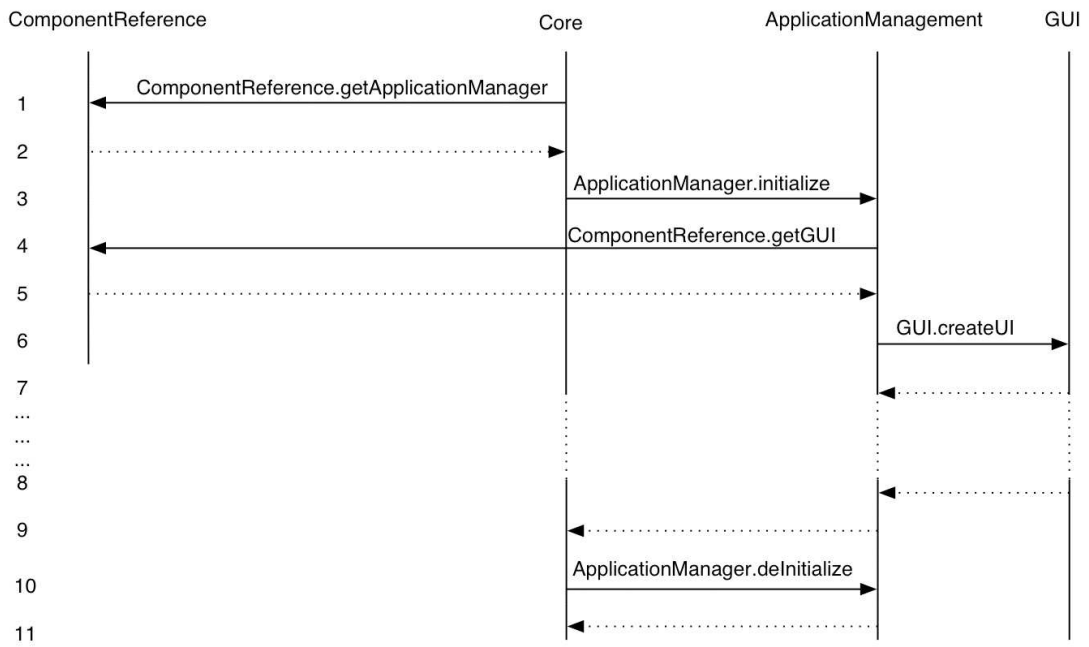
`BowlingGameGUI()`
Luo keilailun käyttöliittymän eli kokoaa käyttöliittymäkohtaiset käyttöliittymäkomponentit.

2.10 Sekvenssikaaviot

Tapahtumien välitys tapahtuu arkkitehtuurissa int-tyyppisten muuttujien avulla. Muuttujassa olevat arvot kuvaavat minkä tyyppinen tapahtuma tapahtui, ja mitä siitä seuraa. Kaikille sovelluksille yleisiä tapahtumia on numeroitu ja nimetty liitteessä 2. Lisäksi sovelluksilla voi olla omia sovelluskohtaisia tapahtumia esiteltynä javadoc-kuvauksessa. Virhetilanteet on esitetty negatiivisin arvoin siten, että virheen sattuessa siitä otettu itseisarvo vastaa sitä tapahtumaa, jota oltiin suorittamassa. Tapahtumien välitystä on kuvattu tarkemmin sekvenssikaavioissa 2.1.4.1, 2.1.4.2 ja 2.1.4.3.

2.10.1 Ohjelman käynnistymisen sekvenssikaavio

Kuva 2.1.4.1 esittää sovelluksen käynnistymisessä tapahtuvat metodikutsut komponenttien välillä. Core on ennen kuvassa esitettyä tilannetta suorittanut ComponentLoader -luokan avulla ComponentReference-luokan (singleton) muodostamisen. ComponentReference-luokka pitää sisällään tiedon sovelluksen erikoistamista komponenteista.



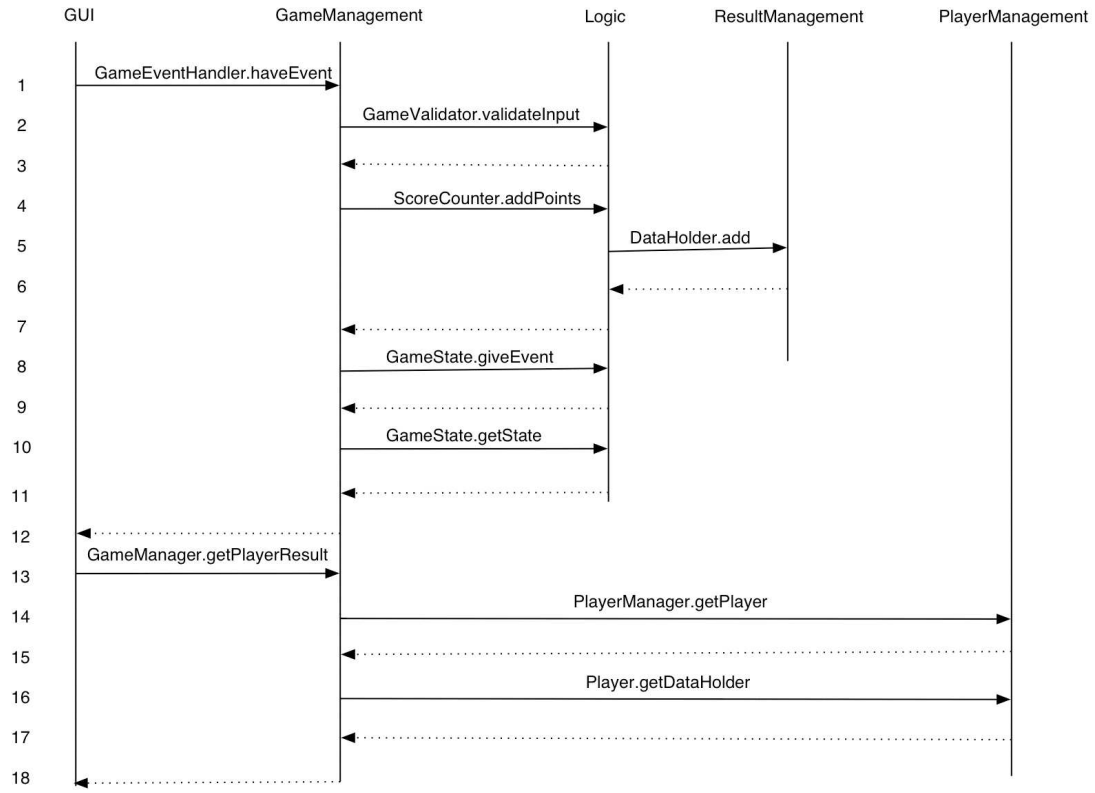
Kuva 2.14: Ohjelman käynnistymisen sekvenssikaavio

1. Core pyytää ComponentReference-luokalta viitettä ApplicationManagement-komponentin ApplicationManager-luokkaan.
2. Kutsu saa paluuarvona viitteen sovelluksen käyttämän ApplicationManager-luokan ilmentymään. ComponentReference luo ApplicationManager-luokan.
3. Core kutsuu ApplicationManager.initialize()-metodia.
4. initialize()-metodi kutsuu ComponentReferenceä saadakseen viitteen sovelluksen käyttämään GUI-komponentin ilmentymään.
5. Koska GUI-komponenttia ei ole vielä olemassa, ComponentReference luo GUI:n ilmentymän, ennen kuin palauttaa viitteen initialize()-metodille.
6. initialize()-metodi kutsuu GUI.CreateUI-metodia, joka käynnistää GUI:n.
7. Metodikutsu palautuu initialize()-metodiin, joka jää odottamaan signaalia GUI-komponentilta, jossa kerrotaan sovelluksen sulkemisesta.
8. GUI-komponentti lähettää sovelluksen sulkemisen merkiksi signaalin ApplicationManagerille.
9. Coren kohdassa 3 tekemä initialize()-kutsu palautuu takaisin Core-luokalle.

10. Core kutsuu ApplicationManager.deInitialize()-metodia sovelluksen hallitun sammuttamisen aloittamiseksi.
11. Kohdassa 10 tehty kutsu palaa Corelle, ja sovellus lopetetaan.

2.10.2 Pisteiden lisäyksen sekvenssikaavio

Kuva 2.1.4.2 esittää, mitä tapahtuu sen jälkeen, kun käyttäjä syöttää sallitun syötteen käyttöliittymän pistekenttään. Tämä sekvenssikaavio on yleinen esimerkki mahdollisesta toteutuksesta. Kuitenkin sovelluskohtaiset toteutukset voivat erota tässä esitetystä mallista. Kaaviossa on esitetty komponenttien välillä tapahtuvat kutsut.



Kuva 2.15: Pisteiden lisäyksen sekvenssikaavio

1. GUI-komponentissa tapahtumakäsittelijä ottaa vastaan käyttäjän syöttämät pisteet. Käyttöliittymässä pisteiden syöttäminen voi tapahtua lisäämällä pisteet tekstikenttiin tai lisäämällä esim. painamalla nappia käyttöliittymässä. Lisäystapa on sovelluskohtainen. Tapahtumankäsittelijä kutsuu GameManagement-komponentin GameEventHandler-luokan haveEvent()-

metodia antaen parametrina tapahtuman tyypin ja viitteen GameEventComponentiin, josta tapahtuma sai alkunsa. Tapahtumat välitetään int-tyyppisinä muuttujina.

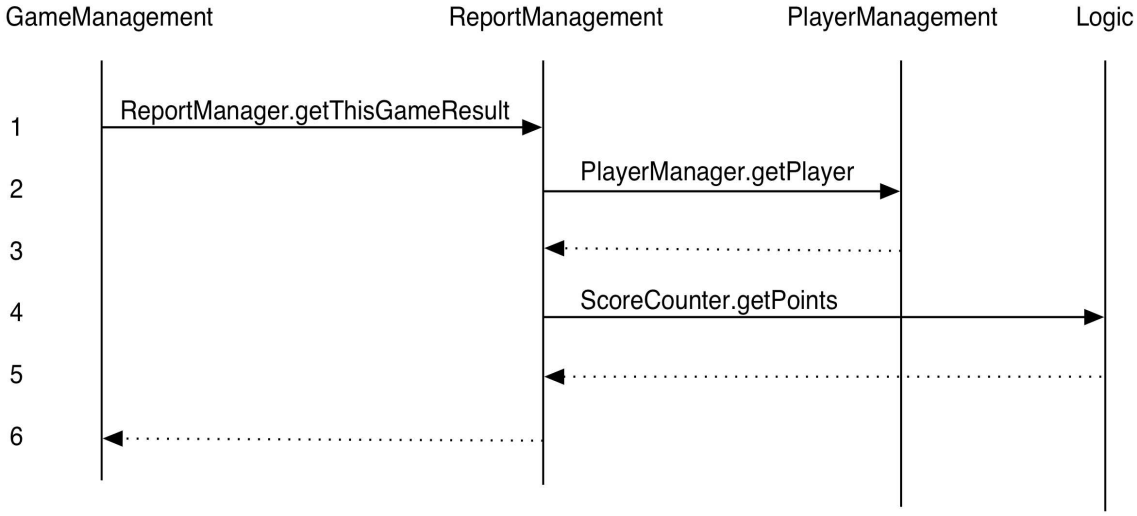
2. haveEvent()-metodi tarkistaa saadun syötteen kutsumalla Logic-komponentin GameValidator.validateInput()-metodia.
3. Logic-komponentin GameValidator.validateInput() antaa paluuarvonaan ”true” tai ”false” riippuen siitä onko annettu syöte sallittu. Jos syöte ei ole sallittu, haveEvent()-metodi välittää negatiivisen arvon takaisin sen kutsujalle, joka käsittelee sen virheenä. Pisteiden syöttämisen yhteydessä se tarkoittaa että annetut pisteet eivät ole sallittuja ja siitä ilmoitetaan käyttöliittymässä virheilmoituksella.
4. haveEvent-metodi() kutsuu vuorossa olleen pelaajan Scorecounter.addPoints() –metodia. ScoreCounter-luokka huolehtii pisteenlaskusta, se hakee kunkin pelaajan dataholderin ja lisää annetut pisteet sinne.
5. ScoreCounter.addPoints()-metodi tallentaa pisteet pelaajan tietorakenteeseen kutsumalla pelaajan DataHolder.add()-metodia.
6. Palataan scoreCounter()-metodiin.
7. Palataan haveEvent()-metodiin.
8. Kutsutaan Logic-komponentin GameState.giveEvent()-metodia, ja annetaan parametrina tapahtuman tyyppi. GameState päivittää sisäisen pelitilanteensa annetun tapahtuman mukaan.
9. Palataan haveEvent()-metodiin.
10. Palataan GUI-komponenttiin ja haveEvent()-metodi antaa paluuarvonaan uuden pelitilanteen.
11. GUI kutsuu GameManagement-komponentin GameManager.getPlayerResult()-metodia saadakseen selville yhden pelaajan syötekenttien uudet arvot.
12. GameManagerissa kutsutaan PlayerManager.getPlayer()-metodia, jotta saadaan viite pelaajaan.
13. Palauttaa viitteen pelaajaan.
14. GameManagerissa pyydetään Player.getDataHolder()-metodilla pelaajalta viite pelaajan pistetietoihin.
15. Palautetaan viite pistetietoihin.

16. GameManager muokkaa pistetiedoista kaksiulotteisen taulukon, joka vastaa pelin GUI:n pistekenttiä. Taulukko palautetaan GUI-komponentille, joka päivittää pistekentät. Joissain peleissä, kuten keilailussa on tarvetta välituloksille, pelin pisteet ja välitulokset välitetään siksi samassa kaksiuloitteisessa taulukossa. Peleissä joissa ei ole välituloksia käytetään taulukosta vain sen piste osaa.

Kohdat 11-16 voivat toistua jokaisen pelaajan kohdalla. Syötekenttien muodostaminen saattaa sovelluksesta riippuen sisältää kutsuja Logic-komponentin ScoreCounter-luokkaan.

2.10.3 Loppuraportin luomisen sekvenssikaavio

Kaaviossa esitetään komponenttien välillä tapahtuvat metodikutsut, kun pelistä luodaan tulosluettelo. Sekvenssikaavio lähtee tilanteesta, jossa GameEventHandler-luokan haveEvent()-metodi on jo saanut käyttöliittymältä pyynnön luoda tulosluettelo pelaajien tuloksista.



Kuva 2.16: Loppuraportin luomisen sekvenssikaavio

1. Kutsutaan ReportManagement-komponentin ReportManager.getThisGameResult()-metodia.
2. ReportManager.getThisGameResult()-metodi kutsuu PlayerManagement-komponentin PlayerManager.getPlayer()-metodia saadakseen viitteen pelaajaan.
3. Saa paluuarvonaan viitteen pelaajaan.
4. ReportManager.getThisGameResult()-metodi kutsuu Logic-komponentin ScoreCounter.getPoints()-metodia.
5. Saa paluuarvonaan yhteispistemäärän pelaajan tuloksista. Kohdat 2-5 toistuvat jokaisen pelaajan kohdalla.
6. Palautetaan tulosluettelo.

Sovellukset tuottavat pelituloksista myös lopputuloslistoja. Näiden tietojen tallentamiseksi käytetään perustoteutuksessa erilaisia tiedostorakenteita.

Tiedostorakenteet on esitelty liitteessä 3. ReportManagement-komponentti huolehtii tuloslistojen muotoilusta ja tallennuksesta.

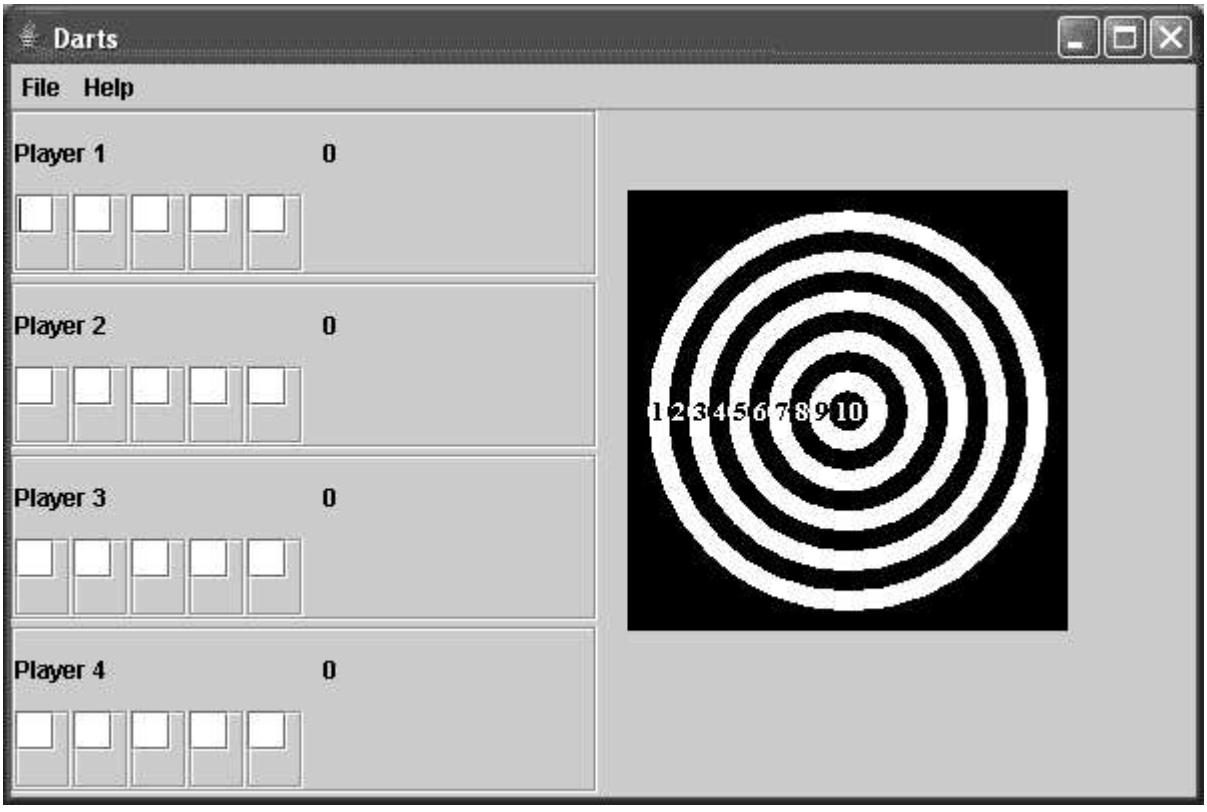
3 Käyttöliittymien ulkoasu

Tässä kappaleessa esitetään käyttöliittymien toiminnat käyttäjän näkökulmasta. Käyttöliittymät tarjoavat peleihin mahdollisuuden syöttää pisteitä sekä katsoa pelin etenemistä ja lopputuloksia (vaatimus 2.1).

3.1 Mökkitikka

Käyttöliittymässä pelaajat syöttävät kierroksen pisteensä käyttöliittymän syöttökenttiin ja ohjelma laskee pisteet automaattisesti yhteen kierroksen tuloskenttään. Pelaajan yhteispisteet näkyvät pelaajan nimen perässä.

Kunkin pelaajan seuraavan heittovuoron syöttökentät aktivoituvat sen jälkeen, kun edellisen kierroksen pisteet on syötetty. Pelin alkaessa kaikkien pelaajien ensimmäisen heittovuoron kentät ovat aktiiviset ja muiden kierrosten kentät eivät ole aktiivisia, ts. niihin ei voi syöttää pisteitä.



Kuva 3.1: Mökkitikan käyttöliittymä

Käyttöliittymä tarkistaa myös syötettyjen pisteiden oikeellisuuden, mökkitikassa syötettävät pisteet ovat numerot 0-10. Jos käyttäjä syöttää virheellisen arvon kenttään, ilmoittaa käyttöliittymä siitä virheilmoituksella heti pisteen syöttämisen jälkeen.

Kuva 3.2: Pisteiden virheilmoitus

3.1.1 Mökkitikan menubar

Mökkitikan menubar tarjoaa kaksi alasvetovalikkoa, jotka ovat:

- File ja
- Help.

File-alasvetovalikko tarjoaa sovelluksen käyttäjälle mahdollisuuden aloittaa uusi peli tai lopettaa sovelluksen käyttäminen.

Kuva 3.3: File-alasvetovalikko

New Game aloittaa uuden pelin kysymällä pelaajien lukumäärän ja nimet JDialog-ikkunan avulla. Toiminta on kaksivaiheinen. Ensin valitaan pelaajien lukumäärä (1 – 10 pelaajaa).

Kuva 3.4: Pelaajien lukumäärän kysyminen

Seuraavassa vaiheessa ikkuna kysyy pelaajien nimet. Ikkuna muokautuu valitun pelaajamäärän mukaan. Syöttökenttiä on siis 1 – 10.

Kuva 3.5: Pelaajien nimen kysyminen

Jos käyttäjä syöttää pelaajan nimikenttään liian pitkän nimen (max 50 merkkiä) tai jättää kentän tyhjäksi, ilmoittaa käyttöliittymä virheestä seuraavanlaisella JDialog-ikkunalla.

Kuva 3.6: Nimen kyselyn virheilmoitus

Käyttöliittymä ei anna käyttäjän jatkaa, ennen kuin kaikkiin pelaajan nimi kenttiin on lisätty oikeanlainen syöte.

Uuden pelin voi aloittaa myös kesken meneillään olevan pelin. Jos New Game -toiminto valitaan kesken toisen pelin, kysyy käyttöliittymä vahvistuksen toiminnolle JDialog-ikkunalla.

Kuva 3.7: New Game -kyselyikkuna

Quit-toiminnolla käyttäjä voi lopettaa sovelluksen käyttämisen. Sovelluksen voi lopettaa myös kesken pelin. Käyttäjän valitessa **Quit**-toiminnon käyttöliittymä tekee tarkistuksen siitä, että valittu toiminto todellakin halutaan suorittaa.

Kuva 3.8: Pelin lopetuksen kyselyikkuna

Help-alasvetovalikosta käyttäjä voi katsoa sovellukseen liittyviä tietoja valitsemalla siitä **About-** toiminnon.

Kuva 3.9: Help-alasvetovalikko

3.1.2 Mökkitikan loppuraportti

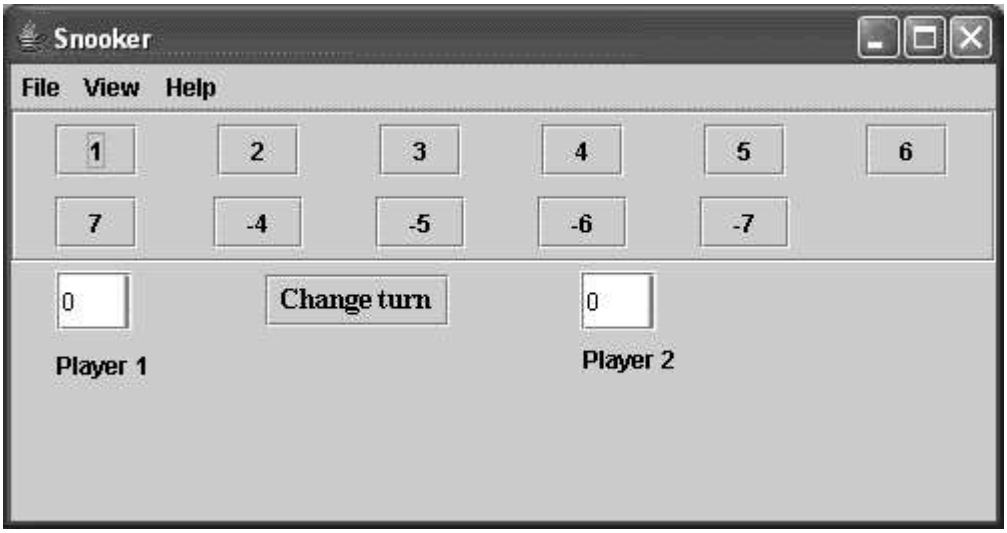
Mökkitikan lopuksi tulostetaan uuteen ikkunaan raportti pelin tuloksesta. Pelaajat listataan pistemäärän mukaisessa järjestyksessä.

Kuva 3.10: Loppuraportti

3.2 Snooker

Snookersovelluksen käyttöliittymällä käyttäjä voi lisätä snookerin pisteitä JButton-napeilla. Yhteispisteet näkyvät pelaajakohtaisessa tekstikentässä. Pelissä tapahtuva vuoronvaihto tapahtuu painamalla käyttöliittymässä **Change turn** -nappia.

Käyttöliittymän pisteidensyöttönapit aktivoituvat pelin edetessä siten, että ne seuraavat snookerin sääntöjä ja logiikkaa. Näin käyttöliittymä estää käyttäjää syöttämästä vääriä pisteitä.



Kuva 3.11: Snookerin käyttöliittymä

3.2.1 Snookerin menubar

Snookerin menubar tarjoaa kolme alasvetovalikkoa, jotka ovat:

- File,
- View ja
- Help.

File-alasvetovalikko tarjoaa sovelluksen käyttäjälle mahdollisuuden aloittaa uusi peli tai lopettaa sovelluksen käyttäminen.

Kuva 3.12: File-alasvetovalikko

New Game aloittaa uuden pelin kysymällä pelaajien nimet JDialog-ikkunan avulla. Pelaajamäärä snookerissa on aina 2 pelaajaa.

Kuva 3.13: Kysyy pelaajien nimet

Jos käyttäjä syöttää pelaajan nimi kenttään liian pitkän nimen (max 50 merkkiä) tai jättää kentän tyhjäksi, ilmoittaa käyttöliittymä virheestä seuraavanlaisella JDialog-ikkunalla.

Kuva 3.14: Nimen kyselyn virheilmoitus

Käyttöliittymä ei anna käyttäjän jatkaa, ennen kuin kaikkiin pelaajan nimi kenttiin on lisätty oikeanlainen syöte.

Uuden pelin voi aloittaa myös kesken meneillään olevan pelin. Jos New Game -toiminto valitaan kesken toisen pelin, kysyy käyttöliittymä vahvistuksen toiminnolle JDialog-ikkunalla.

Kuva 3.15: New Game -varoitusikkuna

Quit-toiminnolla käyttäjä voi lopettaa sovelluksen käyttämisen. Sovelluksen voi lopettaa myös kesken pelin. Käyttäjän valitessa **Quit-**toiminnon käyttöliittymä tekee tarkistuksen siitä, että valittu toiminto todellakin halutaan suorittaa.

Kuva 3.16: Pelin lopetuksen varoitusikkuna

View-alasvetovalikkosta sovelluksen käyttäjä voi selailla aikaisempien pelien parhaita breikkejä eli lyöntivuoroja.

Kuva 3.17: View-alasvetovalikko

Best Break avaa uuden JDialog-ikkunan, jossa tulokset näkyvät.

Help-alasvetovalikosta käyttäjä voi katsoa sovellukseen liittyviä tietoja valitsemalla siitä **About-** toiminnon.

Kuva 3.18: Help-alasvetovalikko

3.2.2 Snookerin loppuraportti

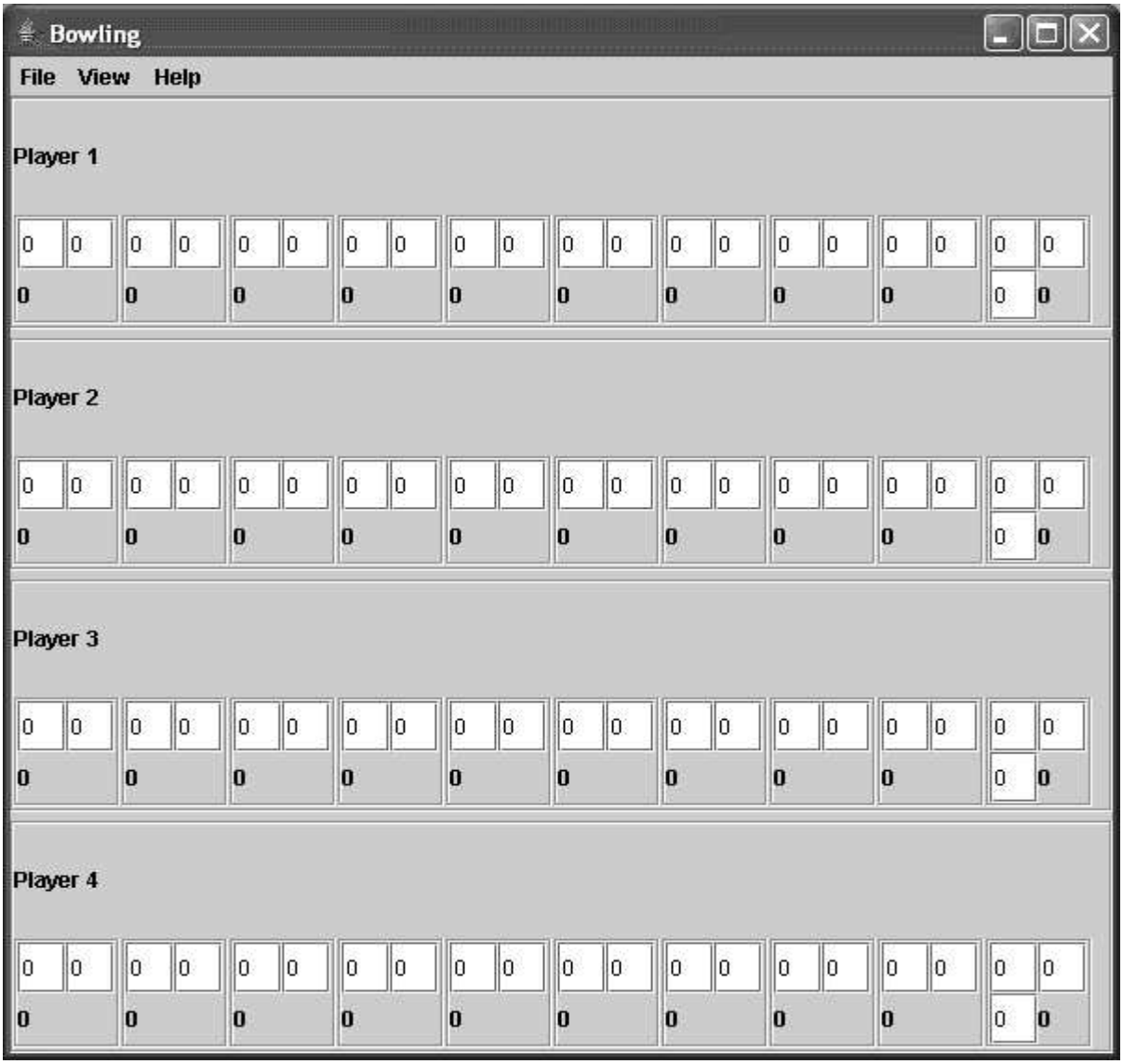
Snooker-sovellus muodostaa pelin päätyttyä käyttäjälle loppuraportin, jossa näkyvät ko. pelin kaksi parasta lyöntivuoroa ja pelaajien nimet. Raportti esitetään JDialog-ikkunassa, josta sovelluksen käyttäjä voi valita **Save-** tai **Close**-toiminnon painamalla JButton-nappia. **Save** tallentaa parhaan lyöntivuoron tulokset ja sulkee ikkunan. **Close** sulkee vain ikkunan tallentamatta.

Kuva 3.19: Snookerin loppuraportti

3.3 Keilailu

Käyttöliittymässä pelaajat syöttävät kierroksen pisteensä käyttöliittymän syöttökenttiin ja ohjelma laskee pisteet automaattisesti yhteen kierroksen tuloskenttään. Pelin kokonaispisteet tulevat näkyviin viimeisen heittokierroksen tuloskenttään.

Kunkin pelaajan seuraavan heittovuoron syöttökentät aktivoituvat sen jälkeen, kun edellisen kierroksen pisteet on syötetty. Pelin alkaessa kaikkien pelaajien ensimmäisen heittovuoron kentät ovat aktiiviset ja muiden kierrosten kentät eivät ole aktiivisia, ts. niihin ei voi syöttää pisteitä.



Kuva 3.20 Keilailun käyttöliittymä

Käyttöliittymä tarkistaa myös syötettyjen pisteiden oikeellisuuden, keilailussa syötettävät pisteet ovat numerot 1-9, /- ja X-merkki. Jos käyttäjä syöttää virheellisen arvon kenttään, ilmoittaa käyttöliittymä siitä virheilmoituksella heti pisteiden syöttämisen jälkeen.

Kuva 3.21: Pisteiden virheilmoitus

3.3.1 Keilailun menubar

Keilailun menubar tarjoaa kolme alasvetovalikkoa, jotka ovat:

- File,
- View ja
- Help.

File-alasvetovalikko tarjoaa sovelluksen käyttäjälle mahdollisuuden aloittaa uusi peli tai lopettaa sovelluksen käyttäminen.

Kuva 3.22: File-alasvetovalikko

New Game aloittaa uuden pelin kysymällä pelaajien lukumäärän ja nimet JDialog-ikkunan avulla. Toiminta on kaksivaiheinen. Ensin valitaan pelaajien lukumäärä (1 – 4 pelaajaa).

Kuva 3.23: Pelaajien määrän kysyminen

Seuraavassa vaiheessa ikkuna kysyy pelaajien nimet. Ikkuna muokkautuu valitun pelaajamäärän mukaan. Syöttökenttiä on 1 – 4.

Kuva 3.24: Pelaajien nimen ja sukupuolen kysyminen

Jos käyttäjä syöttää pelaajan nimi kenttään liian pitkän nimen (max 50 merkkiä) tai jättää kentän tyhjäksi, ilmoittaa käyttöliittymä virheestä seuraavanlaisella JDialog-ikkunalla.

Kuva 3.25: Nimen kyselyn virheilmoitus

Käyttöliittymä ei anna käyttäjän jatkaa, ennen kuin kaikkiin pelaajan nimi kenttiin on lisätty oikeanlainen syöte.

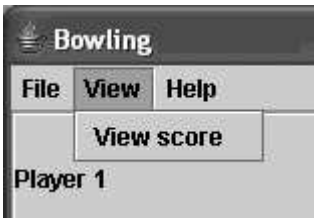
Uuden pelin voi aloittaa myös kesken meneillään olevan pelin. Jos New Game -toiminto valitaan kesken toisen pelin, kysyy käyttöliittymä vahvistuksen toiminnolle JDialog-ikkunalla.

Kuva 3.26: New Game -varoitusikkuna

Quit-toiminnolla käyttäjä voi lopettaa sovelluksen käyttämisen. Sovelluksen voi lopettaa myös kesken pelin. Käyttäjän valitessa **Quit-**toiminnon käyttöliittymä tekee tarkistuksen siitä, että valittu toiminto todellakin halutaan suorittaa.

Kuva 3.27: Pelin lopetuksen kyselyikkuna

View-alasvetovalikkosta sovelluksen käyttäjä voi selailla kolmen edellisen tallennetun pelin tuloksia.



Kuva 3.28: View-alasvetovalikko

View score avaa uuden JDialog-ikkunan, jossa tulokset näkyvät.

Help-alasvetovalikkosta käyttäjä voi katsoa sovellukseen liittyviä tietoja valitsemalla siitä **About-** toiminnon.

Kuva 3.29: Help-alasvetovalikko

3.3.2 Keilailun loppuraportti

Keilailusovellus muodostaa pelin päätyttyä käyttäjälle loppuraportin, jossa näkyvät päivämäärä, pelaajien nimet ja pisteet. Raportti esitetään Jdialog-ikkunassa, josta sovelluksen käyttäjä voi valita **Save-** tai **Close**-toiminnon painamalla JButton-nappia. **Save** tallentaa pelin tulokset ja sulkee ikkunan. **Close** sulkee vain ikkunan tallentamatta.

Raportin ulkoasu on seuraavanlainen:

Kuva 3.30: Keilailun loppuraportti

4 Testaussuunnitelma

Tässä luvussa esitetään projektin testaussuunnitelma. Suunnitelmassa esitetään selvitys kysymyksiin, kuka testaa, mitä, milloin ja kuinka paljon testataan. Testauksen tuloksista kirjoitetaan erillinen dokumentti, jonka vastuuhenkilönä toimii Jarmo Kielosto.

4.1 Testauksen tavoitteet

Testauksen tavoitteena on varmistaa tuotteen korkea laatu, asiakkaan asettamien vaatimusten täyttyminen sekä mahdollisesti dokumentoida ohjelmiston virheet, joita ei ehditä korjaamaan. Tuote koostuu neljästä osasta: ytimestä sekä kolmesta ytimeen pohjautuvasta sovelluksesta. Sovelluksia pyritään testaamaan mahdollisimman yhdenmukaisesti.

4.2 Testauksessa käytettävät apuvälineet ja testiympäristö

Testauksessa käytetään apuna JUnit-testikehystä (www.junit.org) sekä RITA-työkalua (www.cs.helsinki.fi/group/rita/). JUnit-testikehystä käytetään apuna yksikkö- ja integrointitestauksessa. RITA-työkalua käytetään apuna testien riittävän kattavuuden varmistamisessa. Ohjelmiston demonstroitavuus RITA-työkalulla on asiakkaan vaatimus, joten sitä on hyvä kokeilla mahdollisimman aikaisessa vaiheessa, vaikka varsinainen testaus suoritetaan vasta hyväksymistestausvaiheessa. Testausympäristönä on ensisijaisesti Helsingin yliopiston tietojenkäsittelytieteen laitoksen Linux-ympäristö, mutta ohjelmistoa pyritään testaamaan myös Windows-ympäristössä. Java-ympäristöstä on käytössä version 1.4.2 ja RITA-työkalusta versio 0.2.

4.3 Testivaiheet

Testaus jakautuu neljään vaiheeseen:

- yksikkötestaukseen,
- integrointitestaukseen,
- järjestelmätestaukseen ja
- hyväksymistestaukseen.

Yksikkötestit ja integrointitestit toteutetaan JUnit-testeinä siten, että ne ovat ajettavissa RITA-työkalulla. Testauksessa painotetaan yksikkö-, integrointi- ja hyväksymistestausta. Erityisesti käyttöliittymiin keskittyvä järjestelmätestaus on pienemmällä prioriteetilla. Eri vaiheiden testit sijoitetaan eri hakemistoihin, jotka on nimetty testausvaiheen mukaan: yksikkötestit hakemistoon UnitTests ja integrointitestit hakemistoon IntegrationTests.

Yksikkötestauksella ja integrointitestauksella pyritään täydelliseen lausekattavuuteen. Lisäksi integrointitestauksen tavoitteena on testata kaikki toteutettavien sovellusten käyttämät ytimen erikoistamiskohdat.

4.3.1 Yksikkötestaus

Yksikkötestauksessa testataan ohjelmiston pienimmät jakamattomat osat eli luokat. Testaus suoritetaan heti, kun testattava luokka on kirjoitettu. Testit suoritetaan uudestaan aina, kun luokkaan tehdään muutoksia, jotta muutoksista mahdollisesti aiheutuvat virheet huomataan. Yksikkötestauksesta vastaavat Jarmo ja Juha. Testaus toteutetaan lasilaatikkotestauksena (white-box testing), jolloin testattava koodi on näkyvissä. Lasilaatikkotestauksessa testataan ohjelmakoodin rakennetta ja syntaksia. Testauksessa pyritään mahdollisimman kattavaan tulokseen noudattaen seuraavia yleisiä periaatteita:

- kukin koodirivi suoritetaan vähintään kerran,
- kaikki haaraumat testataan,

- silmukat testataan suorittamalla ne seuraavasti: ei kertaakaan, kerran, normaalimäärän, maksimimäärän, kerran yli maksimimäärän.

Yksikkötestit nimetään siten, että testattavan luokan nimen eteen lisätään etuliite Test. Esimerkiksi Player-luokan yksikkötestit ovat luokassa TestPlayer. Kunkin luokan yksikkötestit kootaan yhteen testiluokkaan (JUnitin TestCase). Testiluokista kootaan testijoukot (JUnitin TestSuite) siten, että kaikille sovelluksille yhteiset testit kootaan yhteen testijoukkoon ja sovelluskohtaiset testit omiin joukkoihinsa. Esimerkiksi vain snookeria testaavat testit yhteen joukkoon, snookeria ja mökkitikkaa testaavat yhteen jne. Testitapaukset ja testien tulokset kuvataan testausdokumentissa.

4.3.2 Integrointitestaus

Integrointitestauksessa aiemmin testatut pienemmät yksiköt integroidaan suuremmiksi kokonaisuuksiksi. Testaus suoritetaan mustalaatikkotestauksena (black-box testing), eli testejä laadittaessa keskitytään rajapintoihin eikä integroitavien yksiköiden sisäiseen toteutukseen. Testaus voidaan aloittaa heti, kun yhteenliitettävät luokat on yksikkötestattu. Testausvaiheen lopussa testataan myös komponenttien vaihdettavuutta sovellusten kesken. Testauksesta vastaavat koko projektiryhmä. Erityisesti ytimen erikoistamiskohdat otetaan testien suunnittelussa huomioon.

Integrointitestit kootaan testijoukoiksi samaan tapaan kuin yksikkötestit. Samaa toimintoa testaavat testitapaukset kootaan yhteen testiluokkaan (JUnitin TestCase). Testiluokista kootaan testijoukot (JUnitin TestSuite) siten, että kaikille sovelluksille yhteiset testit kootaan yhteen testijoukkoon ja sovelluskohtaiset testit omiin joukkoihinsa. Testitapaukset ja testien tulokset kuvataan testausdokumentissa.

4.3.3 Järjestelmätestaus

Järjestelmätestauksessa järjestelmää testataan kokonaisuutena. Kutakin sovellusta testataan sen käyttöliittymän kautta. Projektissa toteutetaan kolme

esimerkkisovellusta, jotka kukin testataan erillisinä järjestelminä käyttäen kuitenkin mahdollisimman yhtenäisiä testejä. Testaus suoritetaan ohjelmointivaiheen päätyttyä ja testauksen suorittaa Jarmo.

Testitapaukset on toteutettu määrittelydokumentissa kuvattujen toimintojen ja käyttöliittymäsuunnitelman pohjalta. Testitapaukset on kuvattu liitteessä 4. Testaus todetaan päättyneeksi, kun sovellukset läpäisevät liitteessä kuvatut testitapaukset. Testauksen tuloksista raportoidaan testausdokumentissa.

4.3.4 Hyväksymistestaus

Hyväksymistestauksessa varmistetaan tuoteperheen yhteensopivuus RITA-työkalun kanssa. Asiakas osallistuu hyväksymistestaukseen 7.5.2004.

4.4 Testivaatimusten validointi

Kunkin luokan yksikkötestit kootaan yhteen testiluokkaan. Testiluokista kootaan testijoukot siten, että kaikille sovelluksille yhteiset testit kootaan yhteen testijoukkoon ja sovelluskohtaiset testit omiin joukkoihinsa. Esimerkiksi vain snookeria testaavat testit yhteen joukkoon, snookeria ja mökkitikkaa testaavat yhteen jne.

Integrointitestit jaetaan testijoukkoihin samaan tapaan. Samaa toimintoa testaavat testitapaukset kootaan yhteen testiluokkaan Testiluokista kootaan testijoukot siten, että kaikille sovelluksille yhteiset testit kootaan yhteen testijoukkoon ja sovelluskohtaiset testit omiin joukkoihinsa.

Määrittelydokumentissa sivuutettiin testivaatimusten validointi. Validointi esitetään seuraavassa kaaviossa.

Vaatimus	Validointi
Vaatimus 2.4.1: Ohjelmistoa tulee voida demonstroida RITA-työkalulla.	Asiakkaaseen ollaan yhteydessä ohjelmointi- ja testausvaiheissa. Testataan lopullisesti hyväksymistestauksessa asiakkaan kanssa.
Vaatimus 2.4.2: Ohjelmiston testaus tulee toteuttaa JUnit-työkalulla.	Yksikkö- ja integrointitestaus suoritetaan Junitilla.
Vaatimus 2.4.3: Arkkitehtuuri toteutetaan niin, että samaa testikoodia on mahdollista käyttää eri sovelluksille.	Testit pyritään toteuttamaan siten, että ne ovat ajettavissa kaikille sovelluksille. Sovelluskohtaiset testit erotetaan erillisiin testijoukkoihin.
Vaatimus 2.4.4: Yksikkö- ja integraatiotestit erotetaan toisistaan sekä nimetään selkeästi.	Yksikkötestit nimetään Test-alkuisiksi. Eri testivaiheiden testit sijoitetaan omiin hakemistoihin.
Vaatimus 2.4.5: Kaikki konfiguraatiot ovat testattavissa; myös eri sovelluksien komponentteja yhdistämällä tehdyt sovellukset tulee voida testata.	Testataan integrointitestauksessa.
Vaatimus 2.4.6: Ohjelmistolle toteutetaan testitapauksia, joissa kerrosarkkitehtuuria käydään läpi siten, että kaikki kerrokset tulevat testattua.	Integrointitestauksessa toteutetaan testejä, jotka testaavat komponenttien yhteistoimintaa.
Vaatimus 2.4.7: Testit ryhmitellään testijoukoiksi (test suite) eri ominaisuuksien mukaan.	Testit jaetaan yhteisiin ja sovelluskohtaisiin testeihin.

Kaavio 4.1: Testivaatimusten validointi

LIITE 1: Alustustiedoston rakenne

Sovelluksen erikoistamiskohdat määritellään sovellukselle annettavassa tiedostossa. Tiedoston tulee sijaita siinä hakemistossa, josta sovellus käynnistetään. Tiedoston tulee esitellä erikoistamiskohdat seuraavasti:

GameLogic = fi.cs.helsinki.ohtur6.logic.simplelogic.SimpleLogic

Ensimmäiseksi kerrotaan erikoistamiskohta ja sen arvoksi asetetaan erikoistavan luokan nimi. Luokan nimen tulee sisältää täydellinen polku luokkaan.

Esimerkiksi komponentti Logic erikoistettaisiin SnookerLogic-komponentiksi seuraavasti:

#Tasoon 2
GameLogic = fi.cs.helsinki.ohtur6.logic.simplelogic.SimpleLogic
GameValidator = fi.cs.helsinki.ohtur6.logic.simplelogic.SimpleValidator
GameState = fi.cs.helsinki.ohtur6.logic.simplelogic.SimpleState
#Tasoon 3
SimpleHelper = fi.cs.helsinki.ohtur6.logic.snookerlogic.SnookerHelper

Tiedostossa ei tarvitse esitellä erikoistamattomia kohtia.

LIITE 2: Tapahtumien indeksit

indeksi	nimi	indeksi	nimi
21	NEW_GAME	-21	FAIL_NEW_GAME
30	NUMBER_OF_PLAYERS	-30	FAIL_NUMBER_OF_PLAYERS
31	CREATE_PLAYER	-31	FAIL_CREATE_PLAYER
32	ADD_PLAYER_ATTRIBUTE	-32	FAIL_ADD_PLAYER_ATTRIBUTE
33	REMOVE_PLAYER	-33	FAIL_REMOVE_PLAYER
50	ADD_POINTS	-50	FAIL_ADD_POINTS
51	CHANGE_POINTS	-51	FAIL_CHANGE_POINTS
70	VIEW_REPORT	-70	FAIL_VIEW_REPORT
71	VIEW_PREVIOUS_REPORTS	-71	FAIL_VIEW_PREVIOUS_REPORTS
72	VIEW_TOP10	-72	FAIL_VIEW_TOP10
90	SAVE	-90	FAIL_SAVE
150	RESET_GAME	-150	FAIL_RESET_GAME
200	END_GAME	-200	FAIL_END_GAME
999	EXIT_GAME	-999	FAIL_EXIT_GAME

Indeksien numerointi on toteutettu siten, että samankaltaiset toiminnot alkavat samalla kymmenellä ja negatiivinen indeksi on vastaavan toiminnon virhetapahtuma.

LIITE 3: Tuloslistat

Tuloslistat tallennetaan käyttäen kolmea eri tiedostoa.

Viimeisimmän pelin tuloslista tallennetaan suoraan muokatussa muodossa ”sovelluksennimi.lastgame” tiedostoon. Tiedosto näyttäisi siis sisällöltään esimerkiksi tältä:

```
Mökkitikka

Pelaaja 1       49
Pelaaja 2       48
Pelaaja 3       34
```

Kolme viimeisintä peliä tallennetaan niin ikään käyttäen valmiiksi muokattuja pelilistoja. Pelitulokset erotellaan toisistaan käyttäen `<separator>` merkintää. Tiedoston nimenä on ”sovelluksennimi.last3games”.

```
Mökkitikka

Pelaaja 1       49
Pelaaja 2       48
Pelaaja 3       34

<separator>
Mökkitikka

Pelaaja 1       44
Pelaaja 2       42

<separator>
Mökkitikka

Pelaaja 1       42
Pelaaja 2       41
Pelaaja 3       23
```

HighScore tulokset tallennetaan muodossa, jossa joka toisella rivillä on pelaajan nimi, ja nimiä seuraavalla rivillä on pelaajan tulos. Tiedoston nimenä on ”sovelluksennimi.highscore”

Malli:

```
Pelaaja 1
42
Pelaaja 2
41
Pelaaja 3
23
```

LIITE 4: Järjestelmätestauksen testitapaukset

	Testitapaus	**Odotettu tulos**	**Testattavat sovellukset**
JT-1	**Pelaajien lukumäärän valitseminen**. Käyttäjä valitsee pelaajien lukumäärän.	Sovellus kysyy pelaajatiedot valitulle määrälle pelaajia ja luo kaikille syötekentät.	Mökkitikka Keilailu
JT-2	**Pelaajatietojen syöttäminen**. Käyttäjä syöttää jokaisen pelaajan nimen, ja mahdolliset muut pelaajatiedot.	Jos käyttäjä jättää nimikentän tyhjäksi tai syöttää virheellisen nimen sovellus ilmoittaa virheestä dialogilla, eikä anna jatkaa eteenpäin ennen kuin kenttiin on asetettu oikeanlainen syöte. Kun tiedot on syötetty, kentissä on syötettyjä tietoja vastaavat oikeat alkuarvot.	Kaikki
JT-3	**Pelin aloittaminen**. Käyttäjä valitsee ”File”-valikosta toiminnon ”New game”.	Sovellus kysyy uuden pelin tiedot ja käyttäjä voi aloittaa pisteiden lisäämisen.	Kaikki
JT-4	**Uuden pelin aloittaminen edellisen ollessa kesken.** Käyttäjä valitsee valikosta toiminnon ”New game”, edellisen pelin ollessa vielä kesken.	Sovellus varmistaa toiminnon dialogilla. Jos käyttäjä vastaa ”Yes”, sovellus kysyy uuden pelin tiedot ja käyttäjä voi aloittaa pisteiden lisäämisen. Jos käyttäjä vastaa ”No”, toiminto perutaan ja käyttäjä voi jatkaa käynnissä olleen pelin pisteiden syöttämistä normaalisti.	Kaikki
JT-5	**Pisteiden lisääminen kenttiin mökkitikassa.** Käyttäjä lisää pelaajien pisteitä kenttiin.	Sovellus hyväksyy kenttiin arvot 1-10. Jos pelaaja syöttää virheellisen arvon pistekenttään, sovellus ilmoittaa virheestä dialogilla. Loppupisteet näytetään oikein.	Mökkitikka
JT-6	**Pisteiden lisääminen tikkataululla.** Käyttäjä lisää pisteitä tikkataulun avulla.	Taulusta valittu pistemäärä kirjautuu oikein pelaajan pistekenttään. Testi suoritetaan vain jos kyseinen ominaisuus toteutetaan.	Mökkitikka
JT-7	**Pisteiden lisääminen keilailussa.** Käyttäjä lisää pelaajien pisteitä kenttiin.	Sovellus hyväksyy kenttiin arvot 1-9, sekä merkit ”/” ja ”X”. Jos pelaaja syöttää virheellisen arvon pistekenttään, sovellus ilmoittaa virheestä dialogilla. Välitulokset päivittyvät oikein.	Keilailu
JT-8	**Pisteiden lisääminen snookerissa.** Käyttäjä lisää pelaajien pisteitä painamalla pistemäärää vastaavaa nappia.	Nappuloista ovat aktiivisina vain ne, jotka ovat sillä hetkellä pelitilanteessa mahdollisia. Pisteet päivittyvät oikein.	Snooker
JT-9	**Loppuraportin katsominen.** Käyttäjä syöttää kaikkien pelaajien kaikki pisteet.	Kun kaikki pisteet on syötetty, sovellus esittää erillisessä ikkunassa loppuraportin, jossa pelin tulokset ovat esitetty oikein.	Kaikki
JT-10	**Sovelluksen sulkeminen.** Käyttäjä sulkee ikkunan tai valitsee ”File”-valikosta ”Exit”-toiminnon.	Sovellus varmistaa toiminnon dialogilla. Jos käyttäjä vastaa ”Yes”, ikkuna sulkeutuu ja sovelluksen suoritus päättyy. Jos käyttäjä valitsee ”No”, toiminto perutaan.	Kaikki
JT-11	**Sovelluksen sulkeminen pelin ollessa kesken.** Käyttäjä sulkee ikkunan tai valitsee ”File”-valikosta ”Exit”-toiminnon pelin ollessa kesken.	Sovellus varmistaa toiminnon dialogilla. Jos käyttäjä vastaa ”Yes”, ikkuna sulkeutuu ja sovelluksen suoritus päättyy. Jos käyttäjä valitsee ”No”, toiminto perutaan ja käyttäjä voi jatkaa käynnissä olleen pelin pisteiden syöttämistä normaalisti.	Kaikki
JT-12	**Tulosten tallentaminen.** Käyttäjä painaa loppuraportti-ikkunassa ”Save”-nappulaa.	Sovellus tallentaa oikeat tulokset ja ne on selattavissa sovelluksen uudelleen käynnistyksen jälkeenkin.	Keilailu Snooker
JT-13	**Tallennettujen tulosten selaaminen**. Käyttäjä valitsee ”View”-valikosta toiminnon ”View score”.	Sovellus näyttää listan tallennetuista tuloksista erillisessä ikkunassa, jonka sulkemisen jälkeen käyttäjä voi käyttää sovelluksen muita toimintoja.	Keilailu Snooker

LIITE 5: Vaatimuksiin viittaaminen eri luvuissa

Toiminnalliset vaatimukset	**Miten vaatimus toteutuu**	**Viitataan luvussa**
Vaatimus 2.2.1: pelikirjanpidon perustoiminnot	Sovelluksissa voi syöttää pisteitä, selata ja tallentaa high score -listoja	3
Vaatimus 2.2.2: Ohjelmiston ytimen tulee olla laajennettavissa siten, että siihen voidaan lisätä muita, vaihtoehtoisia palveluita.	Ytimen komponenteissa on ns. hot-spotteja, joita erikoistamalla voidaan ydintä laajentaa.	2.2.1

Laadulliset vaatimukset	**Miten vaatimus toteutuu**	**Viitataan luvussa**
Vaatimus 2.3.1: Käytetään eksplisiittistä ja yhtenäistä esitystapaa.	Käytetään yhtenäistä esitystapaa kaikissa ryhmän tuottamissa dokumenteissa, javadocissa ja koodissa.	2.1.1
Vaatimus 2.3.2: Sovellukset ovat eritasoisia; yksi sovellus toteutetaan mahdollisimman yksinkertaisesti ja yksi selkeästi muita monimutkaisempana..	Eri sovellukset käyttävät ja erikoistavat osittain samoja komponentteja, mutta erikoistavat myös joitakin komponentteja vain niin että ko. komponenttia erikoistaa vain yksi tai kaksi sovellusta.	2.1
Vaatimus 2.3.3: Sovelluskohtaisten eli erikoistettujen komponenttien tulee olla vaihdettavissa toisiin komponentteihin siten, että lopputuloksena saadaan kääntäjästä läpi menevä tuote. Tällaisen tuotteen ei kuitenkaan tarvitse olla toiminnaltaan järkevä.	Ns. ”haamusovellukset” ovat mahdollisia. Ne menevät läpi kääntäjästä, mutta eivät välttämättä ole ajonaikaisesti vakaita.	2.1.1
Vaatimus 2.3.4: Kaikki luokat ovat osa jotakin komponenttia, ja	Kaikki luokat ovat osana jotain package-rakennetta. Yksikään	2.1.2

komponentit eivät saa olla sisäkkäisiä.	luokka ei ole osana kahta eri package-rakennetta.	
Vaatimus 2.3.5: Ydin on mahdollisimman sama kaikilla sovelluksilla, ja kaikki sovelluskohtaiset lisäykset tehdään määriteltyjen hotspottien kautta.	Sovellukset käyttävät ytimen komponentteja sellaisenaan tai erikoistavat vain erikoistamiskohdiksi määriteltyjä komponentin osia.	2.1
Vaatimus 2.3.6: Kerrosarkkitehtuurissa noudatetaan ns. Hollywood-periaatetta (”Don´t call us, we’ll call you”).	Sovelluksen ajonaikana ydin ohjailee sen toimintaa. Ydintä ei käytetä kuten kirjastoa.	2.2
Vaatimus 2.3.7: Ohjelmistoarkkitehtuuri toteutetaan puhtaalla (ei kutsuja kerrosten yli) kolmikerrosarkkitehtuurilla.	Kun peruskomponentista erikoistetaan uusi komponentti, voidaan uuteen komponenttiin lisätä luokkia ja metodeja tarpeen vaatiessa. Jos komponenttia erikoistetaan edelleen uudessa komponentissa, perustuu laajennus tähän komponenttiin eikä peruskomponenttiin.	2.1, 2.8
Vaatimus 2.3.8: Ytimeen toteutetaan mahdollisimman paljon ohjelmistosta. Ytimen toteutuksen tulee painottua siten, että ensimmäiseen kerrokseen tulee mahdollisimman paljon ja kolmanteen kerrokseen mahdollisimman vähän toteutusta.	Ytimen koodi painottuu tasolle yksi, jossa on kaikkein eniten koodia. Poikkeuksena kuitenkin ovat GUI ja GameManagement.	2.1
Vaatimus 2.3.9: Kolmikerroksinen rakenne täytyy toteuttaa joidenkin, muttei välttämättä kaikkien komponenttien osalta.	Kolmikerros rakenne toteutuu GUI- ja Logic-komponenttien osilta.	2.1.3.1, 2.1.3.2, 2.1.3.3, 2.8
Vaatimus 2.3.10: Arkkitehtuuri toteutetaan siten, että jotkut komponentit ovat kaikkien kolmen sovellusten käytössä, jotkut kahden sovelluksen käytössä ja jotkut ovat sovelluskohtaisia.	Arkkitehtuuri sisältää peruskomponentteja joita kaikki sovellukset käyttävät (Core, ApplicationManagement, PlayerManagement), sekä erikoistettuja komponentteja, joita yksi sovellus käyttää.	2.4.1

	Vaatimus ei täyty puhtaasti, koska komponentteja joita käyttää kaksi sovellusta ei komponenttitasolla ole. Luokkatasolla vaatimus toteutuu.	
Vaatimus 2.3.11: Arkkitehtuurissa erikseen nimetyt komponentit toteutetaan vain tyhjänä runkona (skeleton), jotka ovat myöhemmin toteutettavissa (implementoitavissa).	Arkkitehtuurissa toteutetaan DatabaseApplication-Manager ja DatabaseReport tyhjinä runkoina.	2.1.2
Vaatimus 2.3.12: Erikoistaminen voidaan toteuttaa kolmella tavalla: 1) abstrakteilla luokilla, 2) rajapintaluokilla (interface) ja 3) konkreettisen luokan perinnällä. Asiakkaan vaatimus on, että jokaista tapaa käytetään ainakin kerran.	Kaikkia on toteutettu: esimerkiksi GUI komponentissa on rajapinta GUI ja abstrakti luokka GameManager. Logic-komponentissa on luokka SimpleHelper, josta peritään esimerkiksi SnookerHelper.	2.4.1
Vaatimus 2.3.13: Koodi (luokkien, metodien ja muuttujien nimet sekä kommentit) ja käyttöliittymät toteutetaan englanniksi.	Koodissa metodit ja muuttujat ovat englanniksi, samoin javadoc.	Huomautus: tähän vaatimukseen ei viitata.
Vaatimus 2.3.14: Sovellusten on erotuttava toisistaan noin 5-10 kehyksestä erikoistettavan tai käytettävän luokan osalta.	Bowling eroaa kehyksestä seitsemän luokan osalta, Darts eroaa kehyksestä neljän luokan osalta, Snooker eroaa kehyksestä viiden luokan osalta. Nämä kaikki ovat luokkia, joita muut sovellukset eivät käytä. Lisäksi dartsilla ja snookerilla on kaksi yhteistä erikoistettua luokkaa.	2.1
Vaatimus 2.3.15: Sovellusten tulee olla yksinkertaisia, mutta silti niiden pitää sisältää riittävästi variaatiota	Sovellukset on suunniteltu siten, että niissä voidaan luoda monipuolisia testitapauksia, jotka	2.1.2

analysointia varten.	kattavat 1-3 sovellusta.	
Vaatimus 2.3.16: Ohjelmisto ei saa käyttää **java.lang.reflect**-paketin luokkia.	Kyseistä pakettia ei käytetä ollenkaan.	Huomautus: tähän vaatimukseen ei viitata.