JAVA FX GUI -HARJOITUKSET

Tässä dokumentissa olevat Java-harjoituset liittyvät Java FX GUI-ohjelmiin, mikä tarkoittaa että ohjelmissa on Graphical User Interface eli graafinen käyttöliittymä, ja lisäksi nuo graafiset käyttöliittymät on toteutettu Java FX -luokkia hyödyntäen.

Kari Laitinen
http://www.naturalprogramming.com
2015-01-04 Tiedosto luotu.
2015-10-05 Viimeisin muutos

OHJEITA HARJOITUSTEN TEKOON

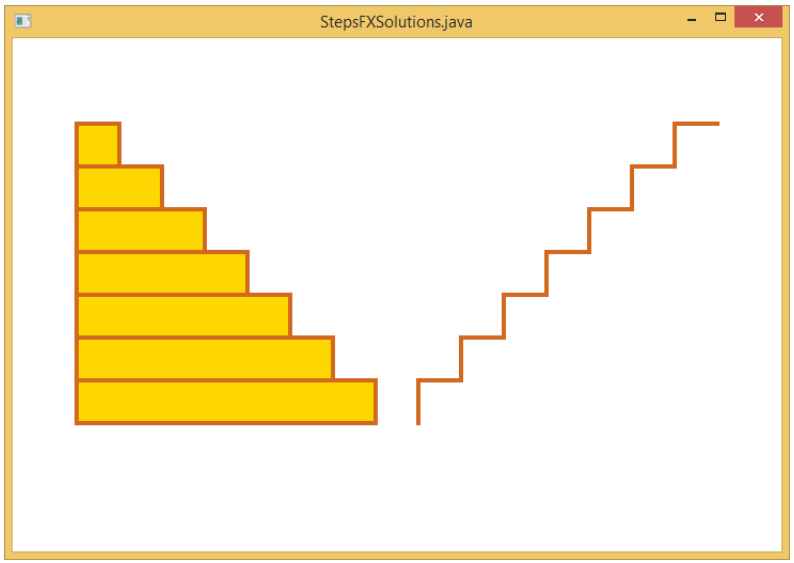
- Nämä harjoitukset ovat pääsääntöisesti sellaisia, että alussa otetaan käyttöön jokin esimerkkiohjelma, jota muutetaan ja edelleenkehitetään harjoituksissa. Jokaiseen harjoitukseen kuuluu yleensä monta osatehtävää. Tarkoitus on että yhdellä harjoituskerralla tehdään aina johonkin yhteen ohjelmaan liittyvät tehtävät.
- JCreatorilla työskenneltäessä ei ole tarpeellista perustaa projekteja. Riittää kun .java tiedostot avataan sellaisenaan JCreatoriin ja tarvittavat tiedostot ovat kaikki samassa kansiossa.
- Varo antamasta Java-ohjelmallesi esim. nimeä String.java, koska String on Javan standardiluokka. Anna ohjelmallesi mieluiten pitkähkö nimi tyyliin KivaPeliHarjoitus.java. Muista myös että Javassa on sääntö jonka mukaan ohjelman pääluokan nimi tulee olla KivaPeliHarjoitus jos tiedostonimi on KivaPeliHarjoitus.java.
- Säädä ohjelmaeditorisi sellaiseksi että tabulointinäppäimen painallus vastaa kolmea välilyöntiä. Tällaista ohjelmointityyliä on käytetty Kari Laitisen esimerkkiohjelmissa joita näissä harjoituksissa muutellaan. Paras on säätää editori sellaiseksi että tabulointimerkin tilalle tiedostoon tulee kolme välilyöntiä. (Yleensä editoreissa on Settings- tai Configure-menu josta tämmöisen säätämisen voi tehdä.)

HARJOITUKSIA OHJELMALLA StepsFX.java

StepsFX.java on ohjelma joka näyttää ikkunassa kahdella tavalla rakennetut graafiset portaat. Portaita on tehty sekä `Rectangle`- että `Line`-olioita käyttäen. Näissä harjoituksissa ’parannellaan’ kyseistä ohjelmaa ja samalla tutustutaan Java FX -sovellusten rakenteeseen.

Tehtävä 1:

Nykyisellään ohjelmassa on nousevat ja laskevat portaat. Muuta ohjelma sellaiseksi että siihen tulee laskevat ja nousevat portaat seuraavan kuvan mukaisesti:

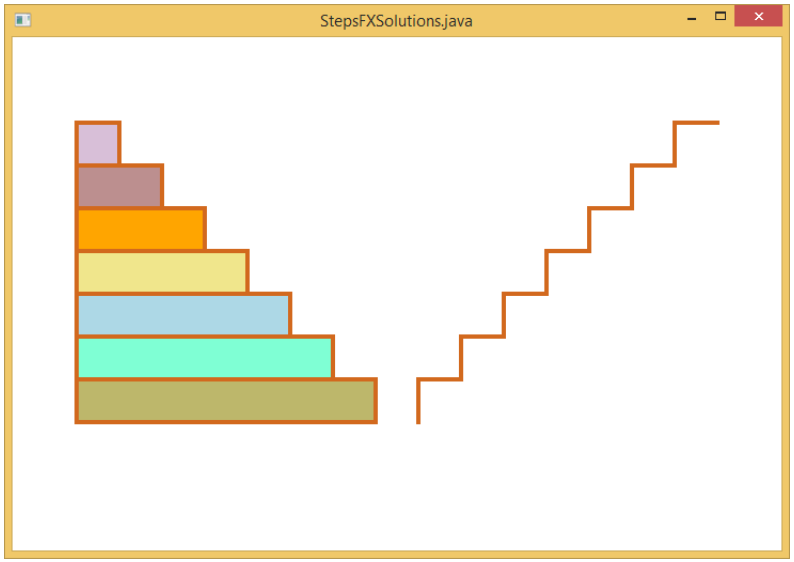


Tehtävä 2:

Tee laskevien portaiden suorakaiteista erivärisiä. Tämän saat aikaiseksi esimerkiksi jos käytät seuraavaa taulukkoa jota voit sopivasti indeksoida siinä silmukassa jossa luot `Rectangle`-olioita:

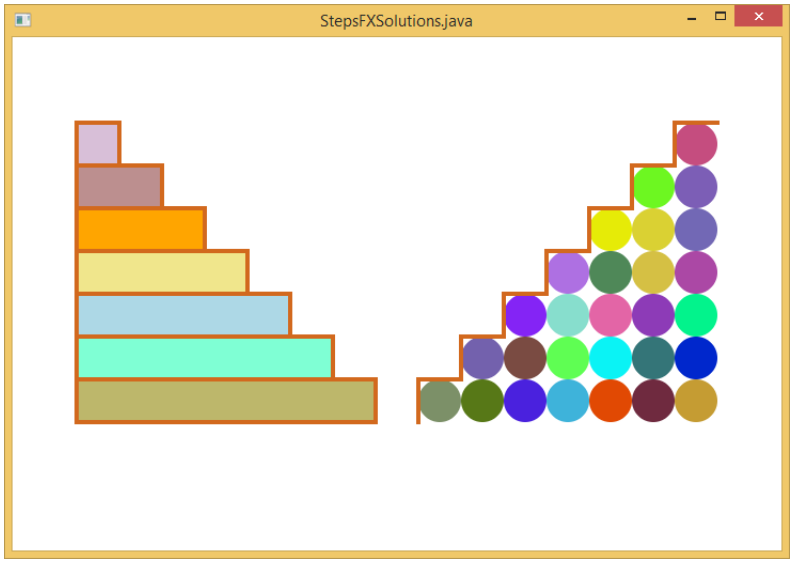
```
Color[] step_colors = { Color.DARKKHAKI, Color.AQUAMARINE, Color.LIGHTBLUE,
                        Color.KHAKI, Color.ORANGE, Color.ROSYBROWN,
                        Color.THISTLE, Color.TOMATO } ;
```

Portaat voivat tämän tehtävän seurauksena näyttää seuraavanlaisilta:



Tehtävä 3:

Laita nousevien portaiden alle perustukseksi erivärisiä palloja, eli `Circle`-olioita, seuraavan kuvan mukaisesti

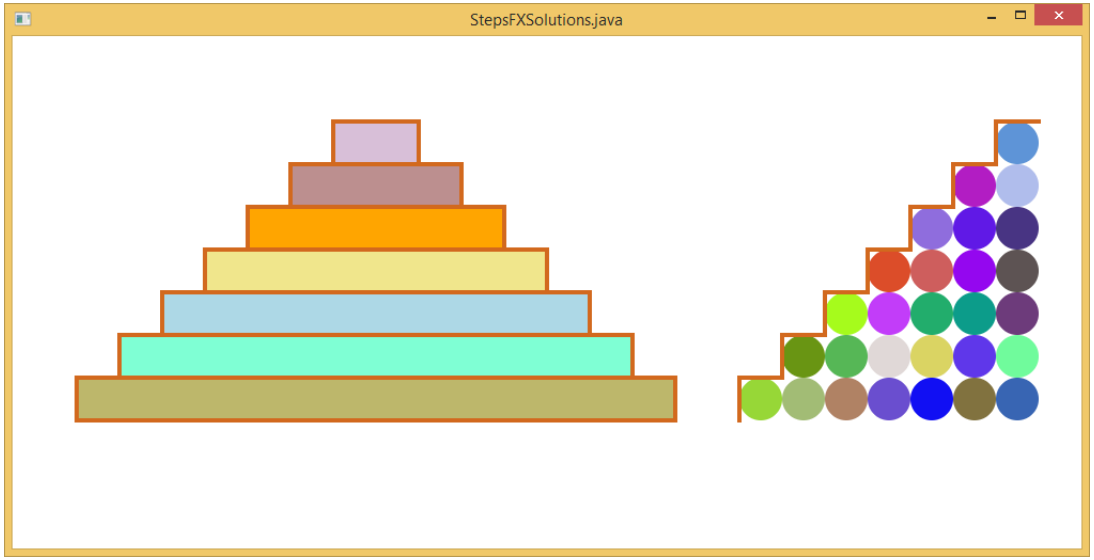


Nuo pallot on mahdollista saada aikaiseksi yhdellä `for`-silmukalla jonka laitat nousevia portaita rakentavan silmukan sisään. Pallojen väriksi voi laittaa satunnaisen värin jonka saa aikaan esim. seuraavalla lauseella.

```
Color random_color = Color.color( Math.random(), Math.random(),
                                  Math.random() ) ;
```

Tehtävä 4:

Muuta vasemmalla olevat laskevat portaat pyramidiksi seuraavan kuvan mukaisesti.



Jotta saat pyramidin mahtumaan ikkunaan, kannattaa ikkunasta (`Scene`-oliosta) tehdä esim. 280 pistettä leveämpi.

Kun olet tehnyt nämä harjoitukset voit sanoa opetelleesi rakentamaan pyramidin jos joku kysyy että mitä olet viime aikoina opiskellut.

HARJOITUKSIA OHJELMALLA SinglePictureFX.java

Ohjelman **SinglePictureFX.java** onnistuneeseen suorittamiseen tarvitset ohjelman käyttämän kuvatiedoston jonka löydät kansiosta

`http://www.naturalprogramming.com/javagui/javafx/`

Harjoitus 1:

Tee ohjelmasta **SinglePictureFX.java** sellainen että kuva näytetään vain yhteen kertaan, sen luonnollisessa koossa, siten että kuva tulee täsmälleen ikkunan keskelle.

Tässä voit käyttää jotain muutakin kuvaa kuin ohjelman alunperin käyttämää kuvaa. Tässä täytyy ottaa selville ohjelman `Scene`n leveys ja korkeus kuten esim. ohjelmassa **ShapesDemoFX.java** on tehty.

Kun kuvaa näytetään sille määritellään kuvan vasemman yläkulman paikka. Joudut siis tekemään laskutoimituksia jotta saat vasemman yläkulman sellaiseen paikkaan että kuva tulee alueen keskelle.

(Huom! Kuvatiedostoja käytettäessä on joskus esiintynyt ongelmia siten että jokin kuva ei vaan suostu näkymään. Kokeile siis jotain toista kuvatiedostoa jos kuvan näkymisessä esiintyy jotain mystisyyttä.)

Harjoitus 2:

Lisää ominaisuus että kuvalle tehdään raamit jollain värillä. Raamit saat kätevästi aikaiseksi ainakin siten että teet hiukan kuvaa suuremman suorakaiteen kuvan alle. Suorakaide tulee täyttää halutulla raamien värillä.

Raamit tekevälle suorakaiteelle pitää laskea sitten sopiva paikka että raamit ovat joka puolella kuvaa saman levyiset.

Harjoitus 3:

Tee ikkunassa näkyvän kuvan taustalle tiiliseinä. Tiiliseinän saat aikaiseksi kun kopioit sopivasti koodia ohjelmasta **BrickWallFX.java**.

Kun tämä osatehtävä on suoritettu, ohjelman pitäisi tuottaa seuraavan kuvan kaltainen näkymä.

Harjoitus 4:

Lisää ohjelmaan ominaisuus että näytettävää kuvaa ja sen raameja voidaan pienentää tai isontaa Nuoli ylös - ja Nuoli alas -näppäimillä. Katso ohjelmasta **KeyboardInputDemoFX.java** kuinka näppäimistöön voidaan reagoida

Eräs tapa ratkaista tämän tehtävän ongelma on sellainen että muutat käytettyjen `ImageView`- ja `Rectangle`-olioiden skaalausta. Skaalausta, jonka oletusarvo on 1.0, voi kasvattaa tai pienentää esim. arvolla 0.05 kun mainittuja nuolinäppäimiä painetaan. Mainittujen luokkien olioille on olemassa skaalauksen lukemiseen ja muuttamiseen käytettävät metodit, joita voi käyttää seuraavaan tapaan

```
frame_rectangle.setScaleX( frame_rectangle.getScaleX() + 0.05 ) ;
```

Harjoitus 5:

Jos aikaa riittää ja intoa piisaa, niin lisää ohjelmaan vielä ominaisuus että näytettävää kuvaa voidaan vaihtaa Nuoli oikealle - ja Nuoli vasemmalle -näppäimillä. Tässä voi toimia niin että alussa ladataan näytettävät kuvat `Image[]` -tyyppiseen taulukkoon, ja pannaan olioviittaaja `picture_to_show` osoittamaan vuorollaan kuhunkin taulukon kuvaolioon. Voi myös olla hyödyksi käyttää `ImageView[]` -tyyppistä taulukkoa, tai pitää kuvaolioita `ArrayList`-taulukossa.

HARJOITUKSIA OHJELMALLA MovingBallFX.java

Ohjelma MovingBallFX.java on esimerkki jossa mm. Button- ja ChoiceBox-luokkien avulla on rakennettu graafinen käyttöliittymä. Painonapeilla voi liikuttaa ikkunassa näkyvää palloa ja ChoiceBox-olion avulla voi muuttaa sen väriä. Itse pallo on Circle-luokan olio.

Tehtävä 1:

Lisää ohjelmaan uusi painonappi joka toimii Reset-nappina siten että sen painamisen jälkeen pallo siirtyy alkuperäisen värisenä alkuperäiselle paikalleen scenen keskelle.

- Tässä täytyy mm. luoda uusi Button-olio ja lisätä se HBox-olioon jossa muutkin painonapit ovat ’kiinnitettyinä’.
- Lisäksi täytyy määritellä mitä silloin tehdään kun Reset-nappia painetaan. Tällöin täytyy modifioida Circle-olion keskipistettä ja filliä eli täyteväriä.

Tehtävä 2:

Katso mallia ohjelmasta RectangleFX.java ja laita MovingBallFX.java-ohjelmaan ScrollBar jolla voi säätää näytettävän pallon kokoa, eli käytännössä sen sädettä.

Tehtävä 3:

Katso vielä lisää mallia ohjelmasta RectangleFX.java laita MovingBallFX.java-ohjelmaan esim. 3 kappaletta RadioButton-olioita joilla voidaan valita pallolle ohut reuna, keskipaksu reuna tai paksu reuna. Circle-olion reunan paksuus voidaan määritellä setStrokeWidth()-metodin avulla.

Tehtävä 4:

Tee pallon liikkumiseen sellainen muutos, että esim. jos palloa siirretään oikealle niin paljon että se katoa kokonaan näkyvistä, niin se tulee muutaman napin klikkauksen päästä esiin vasemmasta ikkunan reunasta alkaen, edelleen oikealle siirtyen.

Tämän saman toiminnon voi toteuttaa kaikkiin palloa siirtäviin painonappeihin.

HARJOITUKSIA OHJELMALLA MovingBallsWithMouseFX.java

Ohjelma MovingBallsWithMouseFX.java näyttää kolmea palloa ikkunassa ja hiirellä on mahdollisuus tarttua noihin palloihin ja liikutella niitä. Ohjelma on rakennettu siten että pallot ovat Circle-luokasta johdetun Ball-luokan olioita.

Harjoitus 1:

Muuta ohjelmaa siten että lisäät siihen neljännen näytettävän Ball-olion. Tämä vaatii ainoastaan yhden lauseen ohjelmaan, ja sillä tuo uusi Ball toimii kuten muutkin pallot.

Harjoitus 2:

Johda (periytä) luokasta Ball uusi luokka nimeltä GradientBall. Voit kirjoittaa GradientBall-luokan esim. Ball-luokan jälkeen ohjelmaan. Tarkoitus on että GradientBall-oliot ovat samanlaisia kuin Ball-oliot sillä poikkeuksella että GradientBall-luokan pallossa on värinä ns. gradient-väri.

GradientBall-luokan konstruktorille voidaan antaa parametrina normaali väri, jonka perusteella konstruktori luo gradient-värin jonka voi asettaa esim. setFill()-metodilla pallon väriksi.

Gradientti väri on sellainen että väri muuttuu tiettyjen sääntöjen mukaan väristä toiseksi. Gradientin värin saa aikaan annetusta väristä esim. seuraavilla lauseilla:

```
Stop[] color_stops = { new Stop(0, Color.WHITE ),
                       new Stop(1, given_color ) } ;

LinearGradient gradient_color = new LinearGradient( 0, 0, 1, 1, true,
                                    CycleMethod.NO_CYCLE, color_stops ) ;
```

Tutki luokkaa LinearGradient jos haluat kokeilla muita gradientteja värejä.

Tässä tehtävässä sinun pitää laittaa GradientBall-olioita näkyville jotta voit testata uuden luokkasi toimivuutta.

Harjoitus 3:

Nykyisessä ohjelmassa on se ongelma että viimeisenä liikutettu pallo ei välttämättä jää päälimmäiseksi jos pallot tulevat ruudulle osittain päällekkäin.

Ongelma johtuu siitä että pallot piirtyvät siinä järjestyksessä kuin ne on laitettu Group-olion ’lapsiksi’. Tämä ongelma voidaan ratkaista siten että kun palloa aletaan liikuttamaan, siis kun hiiren nappi on painettu alas pallon päällä, tehdään niin että kyseinen pallo poistetaan Group-olion lapsien listasta ja lisätään sitten listan loppuun. Näin liikuteltava pallo piirtyy viimeisenä ja päälimmäiseksi.

Tuo Group-olion ’lapsilista’ on tyyppiä ObservableList<Node> ja se toteuttaa List-rajapinnan jossa on metodi remove() jolla voidaan jokin olio poistaa listasta. Vastaavasti on metodi add() jolla olio voidaan lisätä listan loppuun.

Harjoitus 4:

Muuta ohjelma sellaiseksi, että jos hiirellä klikataan pallojen ulkopuolelle ikkunassa ja samanaikaisesti on Control-näppäin alhaalla, ikkunaan laitetaan uusi Ball-olio tuohon klikattuun kohtaan.

Tässä tulee ohjelman ’skeneen’ lisätä reagointi hiiren klikkaukseen, eli tarvitset esim. seuraavalla tavalla alkavaa koodia:

```
scene.setOnMousePressed( ( MouseEvent event ) ->
{
   if ( ball_movement_going_on == false )
   {
      Ball new_ball = null ;

      if ( event.isControlDown() )
      {
         ...
```

Ohjelma MouseDemoFX.java on esimerkki jossa hiireen reagointi on toteutettu ’skeneen’.

Uudelle pallolle pitää saada hiiritoiminnot kuntoon. Voit tehdä sen yksinkertaisesti metodikutsulla

```
set_mouse_activities_for_balls() ;
```

Tämä metodi asettaa aina kaikkien pallojen hiiritoiminnot, mutta se ei haittaa ohjelman toimintaa.

Harjoitus 5:

Lisää ohjelmaan ominaisuus että jos hiirellä klikataan pallojen ulkopuolelle ikkunassa ja samanaikaisesti on Shift-näppäin alhaalla, ikkunaan laitetaan uusi GradientBall-olio tuohon klikattuun kohtaan

HARJOITUKSIA OHJELMALLA PlayingCardsFX.java

Ohjelma **PlayingCardsFX.java** on sellainen että siinä voi DEAL-napilla jakaa pelipöydälle viisi korttia riviin ja yhden yksinäisen kortin. SHUFFLE-napilla voi käytettävän korttipakan sekoittaa. Kortteja voi käännellä klikkaamalla niitä hiirellä.

Ohjelma käyttää `playing_cards_images`-nimisessä kansiossa olevia pelikorttien kuvia. Tuommoinen kansio kuvatiedostoineen täytyy saada paikallisesti tietokoneellesi jotta voit tehdä muutoksia ja testata ohjelmaa. Saat pelikorttien kuvat tietokoneellesi kun kopioit, samasta kansiosta jossa ohjelmakin on, **playing_cards_images.zip** -tiedoston ja purat sen siihen kansioon jossa käännät ohjelmasi lähdekoodia.

Harjoitus 1:

Muuta ohjelma sellaiseksi että kortit ovat valmiiksi ’naamapuoli’ ylöspäin silloin kun kortit jaetaan pelipöydälle. `Card`-luokassa on valmiina metodi jolla kortin voi kääntää siten että kortin maa ja arvo ovat näkyvillä. Tähän muutokseen tarvitaan vain pari koodiriviä.

Harjoitus 2:

Muuta ohjelma sellaiseksi että korttipakka on valmiiksi sekoitettu silloin kun ohjelma käynnistyy. Tähän riittää kun lisäät ohjelmaan yhden koodirivin.

Harjoitus 3:

Muuta ohjelma sellaiseksi että kortit ovat valmiiksi jaettuna ’pelipöydälle’ silloin kun ohjelma käynnistyy, eli ei tarvitse painaa DEAL-nappia korttien saamiseksi näkyville.

Tässä kannattaa menetellä siten että kopioit DEAL-nappiin reagoivat koodirivit ja teet niistä oman metodin jolle voit antaa nimeksi esim. `initialize_game()`. Tätä metodia voit kutsua sitten `start()`-metodin loppupuolella pelin alustamiseksi.

Harjoitus 4:

Muuta DEAL-napin toiminto sellaiseksi että sillä saa jaettua uudet kortit vain niiden korttien tilalle jotka on käännettynä kuvapuoli alaspäin. Ohjelman käyttäjä voi siis kääntää nurinpäin ne kortit joiden tilalle hän haluaa pakasta uudet kortit. (Voit unohtaa tuon ’yksinäisen’ kortin kun jaat uusia kortteja.)

Tässä kannattaa menetellä siten että käyt silmukassa läpi rivissä olevat viisi korttia, ja tutkit mitkä niistä ovat nurinpäin. Nurinpäin olevien korttien tilalle otetaan uudet kortit.

Jotta uusi kortti tulee nurinpäin olevan kortin tilalle, pitää tuon poistettavan kortin paikka kopioida uuteen korttiin. Tarvittavan silmukan alkupuoli voi olla seuraavanlainen:

```
for ( int card_index  =  0 ;
          card_index  <  5 ;
          card_index  ++ )
{
   Card card_in_row = (Card) row_of_cards.getChildren().get( card_index ) ;

   if ( card_in_row.card_is_face_down() )
   {
      Card new_card = card_deck.get_card() ;
      new_card.turn_card_face_up() ;

      double position_for_new_card_x = card_in_row.get_card_position_x() ;
      double position_for_new_card_y = card_in_row.get_card_position_y() ;

      new_card.set_card_position( ...
```

Voit asettaa taulukkoon (listaan) uuden kortin `set()`-metodilla vanhan kortin tilalle indeksimuuttujan määräämälle paikalle.

Harjoitus 5:

Kun olet tehnyt edelliset harjoitukset, ohjelma on sellainen että sen käyttäjä (pelaaja) voi ottaa uusia kortteja pakasta ja parantaa viiden kortin muodostamaa pokerikättä.

Muuta ohjelma sellaiseksi että se muuttaa Scenen taustavärin esim. punaiseksi jos rivissä olevat viisi korttia muodostavat pokeripelin värin, eli kaikki kortit ovat samaa maata. Tämän

tarkistuksen voi tehdä sen jälkeen kun uudet kortit on otettu nurinpäinkäännettyjen tilalle. Tarkistukseen riittää pelkkä yksi `if`-rakenne jossa on monimutkainen ehto.

Koska `Card`-oliot sijaitsevat `Group`-olion ’lapsilistassa’, on kortteihin viittaaminen hiukan työlästä, koska ko. listan olioiden oletetaan olevan `Node`-tyyppiä. Tässä täytyy tyypiksi muuttaa `Card` jotta päästään käsiksi oikeisiin metodeihin. Seuraava `if`-rakenne tutkisi onko listan kaksi ensimmäistä korttia samaa maata:

```
if ( ((Card) row_of_cards.getChildren().get( 0 )).belongs_to_suit_of(
        ((Card) row_of_cards.getChildren().get( 1 )) ) )
{
```

Scenen taustavärin muuttamiseksi ohjelmassa pitää päästä viittaamaan nimeen `scene`. Tämän vuoksi DEAL-nappiin liitettyjen toimintojen määrittely pitää siirtää myöhempään kohtaan ohjelmassa jotta `scene` tulee määritellyksi ennenkuin siihen viitataan.

Harjoitus 6:

Muuta ohjelma sellaiseksi että se ilmoittaa, esimerkiksi keltaisella Scenen taustavärillä jos korttirivin viidessä kortissa on pokeripelin neloset eli neljä korttia joilla on sama `card_rank`.

Tämä tarkistus on hiukan monimutkaisempi kuin edellisen tehtävän vaatima tarkistus. Eräs mahdollisuus on käydä läpi silmukassa mahdolliset korttien numeroarvot alueella 1 ... 13 ja laskea kuinka monta kyseisen arvon omaavaa korttia korttirivissä on. Jos näitä kortteja on löydetty neljä kappaletta, on löydetty neloset ja silmukka voidaan lopettaa. Tässä voidaan

tarvita silmukkaa silmukan sisällä.

Huom! Voit halutessasi tehdä seuraavan harjoituksen ilman että teet tätä harjoitusta.

Harjoitus 7:

Koska ohjelmassa oleva SHUFFLE-nappi on tarpeeton edellisten muutosten jälkeen, tee siitä NEW GAME -nappi jolla luodaan uusi korttipakka, sekoitetaan uusi korttipakka, palautetaan alkuperäinen Scenen taustaväri, ja alustetaan peli harjoituksessa 3 tehdyllä metodilla.

HARJOITUKSIA OHJELMALLA BouncingBallFX.java

BouncingBallFX.java on esimerkki animaation toteuttamisesta. Ohjelmassa on metodi nimeltä `handle()` jota kutsutaan automaattisesti useita kymmeniä kertoja sekunnissa. Tuon metodin avulla pallo saadaan liikkumaan ikkunassa.

Ohjelmassa on seuraavanlainen luokkahierarkia:

- Luokan `Bouncer` oliot osaavat liikkua ja pomppia liikkumisalueella kun `move()`-metodia kutsutaan.
- `Bouncer`-luokalla on alaluokka nimeltä `RotatingBouncer`, jonka oliot osaavat pyöriä liikkuessaan.
- `RotatingBouncer`-luokalla on alaluokka nimeltä `ExplodingBouncer`, jonka oliot osaavat räjähtää ’hiljalleen’ kun räjäytys käynnistetään erityisellä metodilla.

Alkuperäisessä ohjelmassa käytetään nimenomaan `ExplodingBouncer`-oliota jolla on kaikki yllä mainitut ominaisuudet.

Seuraavissa harjoituksissa ei tarvitse muuttaa yllä mainittujen luokkien ohjelmakoodia, lukuunottamatta viimeistä harjoitusta. Seuraavat harjoitukset siis tehdään pääsääntöisesti muuttamalla luokan `BouncingBallFX` ohjelmakoodia.

Harjoitus 1:

Laita aluksi ohjelmaan toinen pomppiva pallo. Tämän saat aikaan kun lisäät `BouncingBallFX`-luokan `start()`-metodiin esim. seuraavat lauseet

```
ExplodingBouncer another_ball = new ExplodingBouncer(
                                new Point2D( SCENE_WIDTH / 2,
                                             SCENE_HEIGHT / 2 ),
                                Color.LIGHTYELLOW,
                                bouncing_area ) ;

group_for_balls.getChildren().add( another_ball ) ;
```

Yllä olevalla tavalla tehtynä toisella pallolla on alussa sama paikka kuin ’vanhalla’ pallolla. Uusi pallo lähtee kuitenkin lentämään eri suuntaan koska pallon suunta arvotaan satunnaisesti.

Jotta saat uuden pallon liikkumaan, tulee `handle()`-metodiin lisätä seuraava kutsu

```
another_ball.move() ;
```

Harjoitus 2:

Muuta ohjelma sellaiseksi että sen käynnistyessä lähtee 10 palloa pomppimaan näytöllä. Tämä voidaan tehdä loogisessa mielessä samaan tapaan kuin toimittiin edellisen kohdan yhden pallon kanssa. Ohjelmassa käytetty `Group`-olion ’lapsilista’ on dynaaminen taulukko johon voidaan kätevästi `add()`-metodilla lisätä useampiakin palloja silmukassa. Näin ollen mainitut 10 palloa saadaan näytölle kun `start()`-metodiin laitetaan seuraava silmukka

```
for ( int ball_counter = 0 ;
          ball_counter < 10 ;
          ball_counter ++ )
{

   ExplodingBouncer ball_to_screen =
             new ExplodingBouncer( new Point2D( SCENE_WIDTH / 2,
                                                SCENE_HEIGHT / 2 ),
                                   Color.LIME,
                                   bouncing_area ) ;

   group_for_balls.getChildren().add( ball_to_screen ) ;
}
```

Jotta mainitut 10 palloa saadaan liikkumaan näytöllä tulee `handle()`-metodiin laittaa esim. seuraavanlainen ’foreach’-silmukka joka käy kaikki lapsilistassa olevat pallo-oliot läpi ja kutsuu niille `move()`-metodia.

```
for ( Node ball_as_node : group_for_balls.getChildren() )
{
   ExplodingBouncer ball_to_move = (ExplodingBouncer) ball_as_node ;

   ball_to_move.move() ;
}
```

Kun olet saanut mainitut 10 palloa liikkumaan ikkunassa, voit poistaa alkuperäisen pallon ja alussa lisäämäsi pallon ohjelmasta. (Voit helposti lisätä palloja lapsilistaan jos tuntuu että pelissäsi on liian vähän palloja.)

Harjoitus 3:

Tee kaikille palloille erilainen pohjaväri. Voit hyödyntää tässä seuraavaa taulukkoa

```
Color[] ball_colors = { Color.GOLD, Color.FIREBRICK, Color.DARKVIOLET,
                        Color.DEEPSKYBLUE, Color.OLIVE, Color.ORCHID,
                        Color.ORANGERED, Color.PEACHPUFF, Color.SNOW,
                        Color.THISTLE } ;
```

Siinä silmukassa jossa luot `ExplodingBouncer`-oliot voit indeksoida yllä annettua taulukkoa ja ottaa siitä erilaisen värin jokaiselle pallolle.

Tämän tehtävän teko ei ole edellytys seuraaavan tehtävän tekemiselle.

Harjoitus 4:

Ohjelmassa käytetyille `ExplodingBouncer`-olioille on käytössä metodi nimeltä `contains_point()`, jolla voidaan tutkia onko jokin piste pallon eli ’pomppijan’ alueen sisällä. Lisäksi on käytössä metodi `explode_ball()`, jolla pallo saadaan ’räjähtämään’.

Tehtäväsi on tässä lisätä hiiritoiminto mukaan ohjelmaan siten että hiirellä jotakin palloa klikattaessa se räjähtää ja hiljalleen katoaa näytöltä. Tässä siis ohjelmasta tulee eräänlainen pallojentuhoamispeli.

Esim. silloin kun hiiren nappi painetaan alas voidaan ’kysyä’ jokaiselta ’lapsilistassa’ olevalta pallolta että sattuiko klikattu piste pallon alueelle. Sitten tuhotaan kyseinen pallo jos klikkaus osui sen alueelle.

Huomaa, että pallon tuhoamiseen riittää kun kutsutaan ko. pallon suhteen `explode_ball()`-metodia. `ExplodingBouncer`-luokassa on jo valmiina automatiikka jolla pallo tuhoutuu hiljalleen sitten kun se on räjäytetty `explode_ball()`-metodilla.

Ohjelman alkuperäisessä versiossa `explode_ball()`-metodia kutsutaan silloin kun painetaan näppäimistön Esc-näppäintä.

Tässä voidaan menetellä siten että hiiritoiminto laitetaan ohjelman ’skeneen’. Tähän toimintoon tarvitaan silmukka joka käy kaikki pallot läpi ja räjäyttää niitä tarvittaessa. Hiiritoiminnon tekevä ohjelmanpätkä voi alkaa seuraavasti:

```
scene.setOnMousePressed( ( MouseEvent event ) ->
{
   double clicked_point_x = event.getSceneX() ;
   double clicked_point_y = event.getSceneY() ;

   for ( Node ball_as_node : group_for_balls.getChildren() )
   {
      ExplodingBouncer ball_to_check = (ExplodingBouncer) ball_as_node ;

      if ( ball_to_check.contains_point( ... // jne

      ...
```

Hiireen reagointi määritellään yllä olevassa koodissa Lambda-lausekkeella. Ole tarkkana että päätät määrittelyn oikein.

Harjoitus 5:

Lisää peliin ominaisuus että se käynnistyy vasta kun Space-näppäintä eli välilyöntinäppäintä painetaan. Tämä on aika helppo tehdä kun määrittelee `BouncingBallFX`-luokkaan datajäsenen

```
boolean game_is_being_played = false ;
```

jolle annetaan arvo `true` sitten kun Space-näppäintä on painettu.

Tarkoitus on että palloja aletaan liikuttelemaan vasta sitten kun yllä mainittu muuttuja on saanut arvon `true`. Pallot kyllä piirtyvät ikkunaan vaikka mainitulla muuttujalla on arvo `false`. Tällöin ne piirtyvät päällekäin keskelle ohjelman piirtoaluetta. Pallot sitten sinkoutuvat eri suuntiin automaattisesti kun Space-näppäintä on painettu.

Yllä mainitun muuttujan arvo tulee tutkia myös hiireen reagoivassa metodissa. Palloja ei saa tuhota ennenkuin peli on alkanut.

Harjoitus 6:

Tee ohjelmaan ominaisuus että kun kaikki pallot on räjäytetty, siinä tuhotaan `Group`-olion lapsilistassa olevat tuhoutuneet pallot ja luodaan pallot uudestaan. Ohjelma voi taas tässä tilanteessa jäädä odottamaan Space-näppäimen painallusta.

Tässä kannattaa pallojen luonti tehdä omaksi metodikseen jota kutsutaan alussa ja sitten kun peli on pelattu loppuun ja kaikki pallot on tuhottu.

Jotta voidaan tutkia onko jokin pallo jo räjähtänyt, tulee `ExplodingBouncer`-luokkaan lisätä seuraava metodi.

```
public boolean is_exploded()
{
   return ( ball_state == BALL_EXPLODED ) ;
}
```

Yllä annettu metodi tarvitaan, koska pallo ei ole räjähtänyt heti kun `explode_ball()`-metodia on kutsuttu. `ExplodingBouncer`-luokassa oleva ’automatiikka’ pitää huolen siitä että räjähtäminen tapahtuu hitaasti ja lopuksi pallo siirtyy `BALL_EXPLODED`-tilaan.

Pallojen tila voidaan tutkia `handle()`-metodissa. Jos kaikki pallot ovat räjähtäneet, luodaan uudet pallot ja pannaan `game_is_being_played` arvoon `false`. Jos kaikki pallot eivät ole räjähtäneet ja peli on käynnissä, liikutellaan palloja normaalisti.