HARJOITUKSIA ANDROID-OHJELMOINTIIN LIITTYEN

Kari Laitinen
http://www.naturalprogramming.com
2012-09-04 Tiedosto luotu.
2013-02-20 SquareBallRectangle-harjoitus lisätty.
2019-08-21 Viimeisin muutos.

HARJOITUKSIA OHJELMALLA SquareBallRectangle

Android-esimerkkiohjelmissa on sovellus nimeltä SquareBallRectangle, joka näyttää neliötä, palloa, tai suorakaidetta sen mukaan mikä kuvio on radiobuttoneilla valittuna. Kyseinen sovellus koostuu seuraavista tiedostoista

main/java/square/ball/rectangle/SquareBallRectangleActivity.java
main/java/square/ball/rectangle/SquareBallRectangleView.java
main/res/layout/activity_square_ball_rectangle.xml

Voit saada sovelluksen toimimaan siten että luot Android Studiolla projektin, annat sille nimeksi SquareBallRectangle, laitat pääluokalle nimeksi `SquareBallRectangleActivity`, ja laitat layout-tiedostolle nimeksi **activity_square_ball_rectangle.xml**. Tärkeää on myös että package-nimeksi tulee `square.ball.rectangle`

Kun olet luonut projektin yllä selostettuja nimeämissääntöjä noudattaen, voit kopioida yllä mainitut tiedostot netin esimerkkiohjelmasta oman projektisi vastaaviin kansioihin. Tiedostoja kopioitaessa voidaan ylikirjoittaa Android Studion valmiiksi tekemät tiedostot.

Harjoitus 1:

Muuta ohjelma sellaiseksi että se piirtää suorakaiteen (rectangle) tilalle kolmion (triangle). Tarkoitus on että .xml-tiedostoon laitetaan näytettäväksi tekstiksi "Triangle".

Kolmion piirtämiseen ei `Canvas`-luokassa ole valmiina metodia, mutta kolmion saa aikaiseksi esim. kun käyttää hyväksi `Path`-luokkaa, joka vastaa Standardi-Javan `GeneralPath`-luokkaa. Tuon luokan käyttöön voit katsoa mallia Android-esimerkkisovelluksesta FlyingArrow. Kolmion piirtämiseen tarvittavan `Path`-olion voi rakentaa esim. seuraavilla lauseilla:

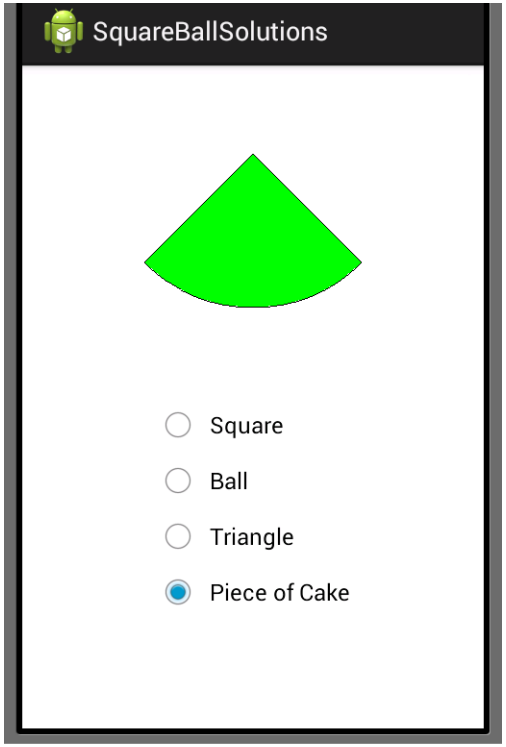
```
Path  triangle_path  =  new  Path() ;

triangle_path.moveTo( view_center_point_x, view_center_point_y - 80 ) ;
triangle_path.lineTo( view_center_point_x + 96, view_center_point_y + 80 ) ;
triangle_path.lineTo( view_center_point_x - 96, view_center_point_y + 80 ) ;
triangle_path.close() ;
```

Kun kolmiota esittävä `Path`-olio on tehty yllä olevalla tavalla, se piirtyy suurinpiirtein keskelle `View`-oliota, eikä koordinaatiston nollapistettä tarvitse tässä siirrellä millään tavalla.

Harjoitus 2:

Muuta ohjelmaa siteen että valittavaksi tulee uusi kuvio nimeltä "Piece of Cake". Tätä varten tulee **.xml**-tiedostoon määritellä uusi RadioButton. Tuon RadioButtonin saat aikaiseksi pelkästään **.xml**-tiedostoa muuttamalla, ja voit testata ohjelmaasi tuon muutoksen jälkeen.

Tässä on tarkoitus että kun valitaan "Piece of Cake", ohjelma tuottaa suurinpiirtein seuraavan kuvan kaltaisen näkymän



Tuollaisen "Piece of Cake"-kuvion saa piirretyksi esim. `Canvas`-luokan `drawArc()`-metodin avulla. Tuon metodin käyttöön löydät mallia DrawingDemo-sovelluksen **DrawingDemoActivity.java**-ohjelmasta.

Androidin `drawArc()`-metodia käytettäessä kulmia lasketaan myötäpäivään. (Monissa muissa järjestelmissä niitä lasketaan vastapäivään.) Huomaa myös että Androidissa esim. suorakaide määritellään parameterein "vasen laita", "ylälaita", "oikea laita" ja "alalaita". Ei siis määritellä suorakaiteen leveyttä ja korkeutta.

Harjoitus 3:

Tässä osatehtävässä on tarkoitus tehdä ohjelmasta oikeaoppinen Android-sovellus. Android-sovelluksen näyttämiä käyttöliittymän tekstejä ei saisi laittaa layout-tiedostoon kuten tiedostossa **res/layout/activity_square_ball_rectangle.xml** on tehty. Oikeaoppisesti tekstit pitää laittaa tiedostoon **res/values/strings.xml**. Tuo tiedosto on jo tällekin sovellukselle olemassa.

Tehtäväsi on nyt siirtää tekstit "Square", "Ball", jne. tuonne **res/values/strings.xml**-tiedostoon ja muuttaa **activity_square_ball_rectangle.xml**-tiedosto sellaiseksi että se osaa viitata mainitussa **strings.xml**-tiedostossa oleviin stringeihin eli merkkijonoihin. Tässä tehtävässä kannattaa katsoa mallia ButtonDemo-sovelluksesta, jossa on oikeaoppisesti tekstit laitettu **strings.xml**-tiedostoon.

Tämän osatehtävän tekeminen ei muuta sovelluksen toimintaa millään tavalla. Voit kuitenkin muuttaa jotain stringiä pikkuisen että testatessasi näät että ohjelmasi toimii. Käytä kuitenkin englanninkielisiä stringejä tässä. Java-tiedostoihin ei tässä tarvitse tehdä muutoksia.

Harjoitus 4:

Tässä osatehtävässä lisätään sovellukseen mahdollisuus suomenkielisten käyttöliittymätekstien käyttöön. Kun olet tehnyt edellisen osatehtävän oikein, voit helposti lisätä suomenkieliset käyttöliittymätekstit kun ensin teet projektin `res`-kansioon alikansion nimeltä `values-fi`. Tuon kansion voi tehdä esim. Windowsin File Explorerilla, tai sitten jotenkin Android Studiolla. Kun tuo uusi kansio (folder) on olemassa, voit kopioida `values-`kansiossa olevan **strings.xml**-tiedoston tuohon uuteen kansioon ja muuttaa siihen suomenkieliset tekstit.

Kun sovellus on emulaattorissa tai oikeassa Android-laitteessa, se osaa automaattisesti käyttää suomenkielisiä tekstejä kun puhelimen käyttökieli muutetaan suomen kieleksi. Emulaattorissakin tämän voi tehdä. Kielen muuttaminen tapahtuu Settings-valikon kautta.

Tässäkään osatehtävässä ei tarvitse Java-tiedostoihin tehdä muutoksia.

Kun itse testasin tätä emulaattorissa, sattui eteen sellaisia vaikeuksia, että kieli ei sovelluksessa muuttunut ennenkuin pysäytin sen Settings-valikon kautta.

Lisää käyttöliittymäkieliä voidaan Android-sovellukseen laitaa kun yllä kuvattuun tapaan luodaan uusia kansioita `values-es`, `values-de`, jne., ja kuhunkin kansioon laitetaan kyseisen kielen mukaiset stringit.

Harjoitus 5:

Jos vielä intoa piisaa, niin voisit kokeilla minkälaista on Kotlin-ohjelmointi.

Kotlin on uusi ohjelmointikieli joka on käytettävissä Android-sovellusten rakentamiseen Android Studion versio 3:ssa.

Näissä harjoituksissa käytetystä sovelluksesta on myös olemassa versio **SquareBallRectangleKt**, joka löytyy **in_kotlin**-nimisestä alikansiosta. Jotta saat tuon sovelluksen käyttöösi pitää sinun perustaa uusi projekti, jossa alussa valitset Kotlinin käytettäväksi. Package-nimi ja pääluokan nimi ovat samat kuin Java-projektissakin.

Projektin luotuasi voit kopioida netistä projektiisi seuraavat tiedostot:

main/java/square/ball/rectangle/SquareBallRectangleActivity.kt
main/java/square/ball/rectangle/SquareBallRectangleView.kt
main/res/layout/activity_square_ball_rectangle.xml

Voit tehdä kokeeksi esim. ensimmäisen harjoituksen Kotlin-versioon. Tämän jälkeen voit sanoa jo omaavasi kokemusta Kotlin-ohjelmoinnista.

Varoitettakoon vielä että Kotlinia käytettäessä Android Studio todennäköisesti herjaa siitä seikasta että Kari Laitisen ohjelmissa on käytetty alaviivaa nimissä.

Harjoitukset ovat jatkossa Java-kieleen perustuvia.

HARJOITUKSIA OHJELMALLA MovingBall

Esimerkkiohjelmista löytyy Android-sovellus nimeltä MovingBall, jonka pääluokka on nimeltään `MovingBallActivity`, joka sijaitsee tiedostossa **MovingBallActivity.java**.

Kyseiseen esimerkkiohjelmaan kuuluu seuraavat tiedostot jotka sijaitsevat projektin tiedostohierarkiassa seuraavasti:

java/moving/ball/MovingBallActivity.java
java/moving/ball/MovingBallView.java
res/layout/activity_moving_ball.xml
res/menu/color_selection_menu.xml
res/values/strings.xml

Nämä kaikki tiedostot on kopioitava perustettavaan projektiin jotta sovellus saadaan toimimaan kehitystyökaluilla. Uusin Android Studio ei tee automaattisesti **menu**-kansiota. Voit tehdä **menu**-kansion itse **res**-kansion alikansioksi kun kopioit tiedoston **color_selection_menu.xml.**

Projektia perustettaessa on laitettava package-nimeksi `moving.ball` ja pää-activityn nimeksi `MovingBallActivity` ja layout-nimeksi `activity_moving_ball`. Uusin Android Studio ei ehkä anna käyttäjän asettaa pää-activityn nimeä tai layout-nimeä projektia luotaessa. Tällöin täytyy muuttaa **AndroidManifest.xml**-tiedostoa ja kirjoitaa `.MovingBallActivity`

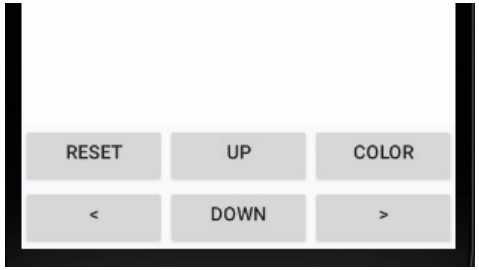
korvaamaan nimi `.MainActivity`

Kun olet saanut sovelluksen toimimaan, tee seuraavat harjoitukset. Huomaa että COLOR-nappia pitää painaa pitkään jotta menu ilmestyy näkyviin.

Harjoitus 1:
Lisää ohjelman käyttämään menuun uusi valittava värivaihtoehto.

Harjoitus 2:
Lisää ohjelmaan Reset-nappi, jolla pallo saadaan takaisin siihen paikkaan ja sen väriseksi kuin se on ohjelman käynnistyessä.

Kun lisäät sovellukseen uuden painonapin, muuta nappien sijoittelu seuraavanlaiseksi.



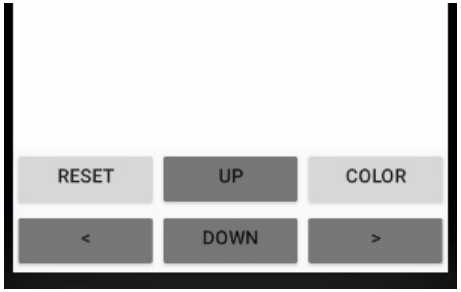
Voit uudelleensijoitella napit muokkaamalla tiedostoa **activity_moving_ball.xml**. Attribuutilla `android:layout_weight` voidaan määrätä kuinka paljon tilaa varataan `LinearLayout`in sisältä. Alkuperäisessä tiedostossa arvot "0.9" ja "0.1" määräävät että 90% tilasta varataan `MovingBallView`-oliolle ja 10% varataan `LinearLayout`-oliolle joka sisältää

painonappeja.

Attribuutille `android:layout_weight` annettavien arvojen ei tarvitse välttämättä sisältää desimaalipistettä. Attribuutille `android:layout_height` pitänee antaa arvo "0dp" silloin kun `android:layout_weight` on käytössä.

Harjoitus 3:

Paranna käyttöliittymää siten että annat eri taustavärin niille painonapeille joita käytetään pallon liikutteluun:



Sovellusta (appia) ButtonDemo tutkimalla saat selville erään keinon painonapin värin asetukseen.

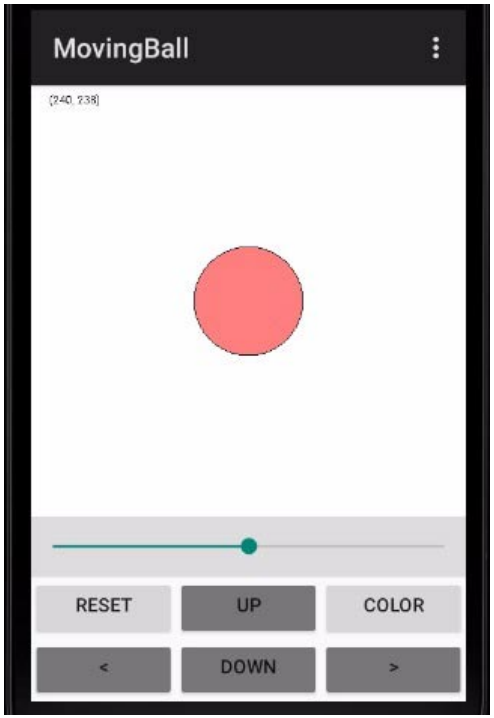
Harjoitus 4:

Muuta sovellus sellaiseksi että pallon säde on alussa täsmälleen 1/8 viewin leveydestä, ja että pallo ’hyppää’ automaattisesti alkuperäiseen keskikohtaan jos sitä liikutetaan siten että se törmää johonkin neljästä ’seinästä’.

Harjoitus 5:

Tutkimalla sovellusta nimeltä TurningArrow saat selville kuinka `SeekBar`-oliota käytetään. `SeekBar`in avulla on helppo muuttaa kokonaisluvun arvoa.

Nyt tehtäväsi on lisätä `SeekBar` MovingBall-sovellukseen siten että sen käyttöliittymä näyttää seuraavanlaiselta



`SeekBar`illa pitää voida säätää pallon värin transparenttisuutta eli läpinäkyvyyttä. Värin läpinäkyvyys voidaan määritellä ns. alfa-arvolla. Androidin `Paint`-luokassa on metodi nimeltä `setAlpha()` jolla alfa-arvo voidaan asettaa.

Alfa-arvo voi olla välillä 0 ... 255, mikä tarkoittaa liikkumista täysin läpinäkyvän ja täysin peittävän värin välillä. Sovelluksen pitää muuttaa pallon täyttövärin alfa-arvoa aina kun `SeekBar`in ’progress’-arvo muuttuu. Alfa-arvo voi olla alussa 128.

Kun olet lisännyt `SeekBar`-olion sovellukseen, sinun täytyy varmistaa että Reset-nappi toimii oikein. Reset-operaatiossa pallon värin alfa-arvo tulee asettaa alkuperäiseen arvoon ja `SeekBar`in ’thumb’ pitää laittaa alkuperäiseen positioonsa.

HARJOITUKSIA OHJELMALLA GesturesDemo

Androidissa on luokka nimeltä `GestureDetector`, jonka avulla voidaan saada selville minkälaisia ’eleitä’ (gestures) käyttäjä tekee kosketusnäytön avulla. Homma toimii siten että tavalliset kosketusnäyttötapahtumat syötetään `GestureDetector`-oliolle, joka generoi niistä eletapahtumia joita ovat

- onDown: näyttöä kosketetaan
- onShowPress: näyttöä kosketetaan vähän kauemmin
- onLongPress: näyttöä kosketetaan vieläkin pidempään
- onScroll: sormella scrollataan
- onFling: sormella pyyhkäistään nopeasti
- onSingleTapUp: sormi otetaan pois näytöltä

GesturesDemo on ohjelma jonka avulla voidaan tutkia näitä eletapahtumia ja nähdään milloin niitä generoidaan. Esim. onSingleTapUp ei tapahdu silloin jos onLongPress on tapahtunut.

Tutki ja koekäytä ensin tätä ohjelmaa ja tee sitten seuraavia harjoituksia. Voit säilyttää ohjelman alkuperäiset ominaisuudet sillä niistä voi olla apua kun lisäät uusia ominaisuuksia ohjelmaan.

Harjoitus 1:

Muuta ohjelma sellaiseksi, että se näyttää pallon siinä kohdassa näyttöä jota painetaan pitkään. Tämä tarkoittaa että `onLongPress()`-metodissa luodaan pallo-olio siinä tapauksessa kun sitä kutsutaan. Voit kopioida valmiin `Ball`-luokan ohjelmasta **MovingBallsWithPointerActivity.java**, ja sijoittaa tuon `Ball`-luokan tiedostoon ennen muita luokkia.

`Ball`-oliota varten tarvitaan `GesturesDemoView`-luokkaan datajäseniksi olioviittaaja joka voidaan määritellä esim. seuraavasti:

```
Ball ball_on_screen ;
```

`Ball`-olio voidaan luoda `onLongPress()`-metodissa esim. seuraavaan tapaan

```
ball_on_screen =
   new  Ball( (int) first_down_motion.getX(),
              (int) first_down_motion.getY(),
               Color.RED ) ;
```

Kun `Ball`-olio piirretään `onDraw()`-metodissa tulee huomioida että olio on todella luotu ennenkuin sitä ryhdytään piirtämään:

```
if ( ball_on_screen != null )
{
   ball_on_screen.draw( canvas ) ;
}
```

Harjoitus 2:

Paranna edellä tehtyä ominaisuutta siten että näytölle ilmestyvälle pallolle annetaan Ball-oliota luotaessa satunnainen väri.

Androidissa värit voidaan ilmaista int-tyyppisinä ARGB-arvoina jotka ovat muotoa 0xAARRGGBB, missä ensimmäinen tavu on ns. alpha-arvo jonka täytyy olla 0xFF jotta saadaan täysin peittävä väri.

Näin ollen satunnainen väri voidaan arpoa kirjoittamalla

```
(int) ( Math.random() * 0xFFFFFF ) + 0xFF000000
```

Harjoitus 3:

Muuta ohjelma sellaiseksi että näytölle ilmestyy aina uusi pallo kun tehdään ’pitkä’ painallus. Tässä siis täytyy luoda useita `Ball`-olioita. Ne kannattaa tallettaa `ArrayList`-pohjaiseen taulukkoon, joka voidaan määritellä esim. seuraavasti:

```
ArrayList<Ball> balls_on_screen = new ArrayList<Ball>() ;
```

Tämmöisessä taulukossa olevat `Ball`-oliot voidaan kätevästi piirtää ’foreach’-silmukalla seuraavaan tapaan:

```
for ( Ball ball_to_draw : balls_on_screen )
{
   ball_to_draw.draw( canvas ) ;
}
```

Harjoitus 4:

Muuta ohjelma sellaiseksi että `onFling()`-operaatiolla on mahdollista tuhota `Ball`-olio. Tämä tarkoittaa että `onFling()`-metodiin kirjoitetaan ohjelmakoodi joka tuhoaa `ArrayList`-taulukosta sen `Ball`-olion jonka kohdalta ’flingaus’ aloitettiin.

Tässä tehtävässä kannattaa käydä `ArrayList`-taulukko läpi lopusta alkaen koska lopussa on uusimmat ja mahdollisesti päälimmäisiksi piirtyvät `Ball`-oliot. `Ball`-luokassa on valmiina metodi `contains_point()` jolla voidaan tutkia sattuuko jokin piste pallon alueelle. `Ball`-luokkaa näissä harjoituksissa ei tarvitse muuttaa. Tässä on hahmotelma ohjelmakoodista joka tutkisi taulukkoa lopusta alkaen:

```
Point first_touched_point = new Point( /* tänne jotain */ ) ;
int ball_index = balls_on_screen.size() ;
boolean ball_to_delete_is_found = false ;

while ( ball_index > 0 && ball_to_delete_is_found == false )
{
   ball_index -- ;

   if ( balls_on_screen.get( ball_index ).
                              contains_point( first_touched_point ) )
   {
      // Tässä kohden pitäisi pallo tuhota taulukosta

      ball_to_delete_is_found = true ;
   }
}
```

Harjoitus 5:

Tee ohjelmaan vielä ominaisuus että kaikille palloille arvotaan uudet satunnaiset värit siinä tapauksessa jos näyttöä ’taputetaan’ nopeasti kolme kertaa esim. 1.5 sekunnin sisällä.

Tämä ominaisuus voidaan toteuttaa `onSingleTapUp()`-metodiin jota systeemi kutsuu silloin kun nopea näytön kosketus loppuu. `MotionEvent`-luokassa on metodi `getEventTime()`, jolla saadaa millisekunteina kasvava arvo tapahtumahetkestä. Kun otetaan talteen kolmen viimeisimmän `onSingleTapUp()`-tapahtuman ajat, on mahdollista laskea milloinka nämä tapahtumat osuvat esim. mainitun 1.5 sekunnin sisään.

Tämän ominaisuuden voi toteuttaa monella tavalla. Eräs mahdollisuus on käyttää `ArrayList`-taulukkoa johon talletetaan aikaa kuvaavia long-arvoja:

```
ArrayList<Long> single_tap_up_event_times =
                                new ArrayList<Long>() ;
```

Pitää odottaa että taulukkoon tulee kolme aika-arvoa ennenkuin aletaan tutkimaan ovatko ne riittävän lähellä toisiaan. Toisaalta taulukon alusta voidaan tuhota turhia arvoja siten että taulukossa on aina vain kolme viimeisintä aika-arvoa, jolloin voidaan verrata aina taulukon ensimmäistä ja viimeistä arvoa. (Alkuperäisessä ohjelmassa säilytetään tietty määrä tekstirivejä `ArrayList`-taulukossa, mistä saattaa saada vinkkiä tämän ominaisuuden toteutukseen.)

HARJOITUKSIA OHJELMALLA AlarmClock

Esimerkkiohjelmien joukosta löytyy AlarmClock-sovellus, joka on herätyskello joka käy hyvin ja siihen voi herätysajankin asettaa. Kyseisessä ohjelmassa on se puute että se ei osaa herättää asetettuna herätysaikana. Muun muassa tämä puute on tarkoitus korvata näissä harjoituksissa.

AlarmClock-sovellus vaatii toimiakseen tiedostot:

src/alarm/clock/AlarmClockActivity.java
src/alarm/clock/AlarmClockView.java
res/layout/alarm_clock_activity.xml

Harjoitus 1:

Ohjelmassa on jo olemassa Activate Alarm -nappi, mutta se ei nykyisellään aiheuta mitään toimintoa. Tee tuolle napille toiminto että kello menee hälytystilaan silloin kun kyseistä nappia painetaan. AlarmClockView-luokkaan tulee tehdä metodit hälytyksen aktivointiin ja deaktivointiin, ja lisäksi sinne tulee laittaa jokin boolean-muuttuja joka ilmoittaa milloin hälytys on aktivoitu. Kellon tulee sitten hälyttää kun hälytys on aktivoitu ja kellon aika on sama kuin asetettu hälytysaika.

Hälytys voi yksinkertaisimmillaan olla näytöllä näkyvä teksti "Herää jo!". Parempi hälytys on jo tuo teksti vilkkuu hälytyksen ollessa päällä. Vielä parempi hälytys on luonnollisesti

äänellä tapahtuva hälytys.

Hälytys pitää saada loppumaan kun Activate Alarm -nappia painetaan uudestaan. Tässä kannattaa ensin rakentaa hälytysominaisuus ja vasta sitten ohjelmoida sen lopettaminen.

Activate Alarm -napin taustavärin voi pistää esim. punaiseksi ja tekstinkin voi vaihtaa silloin kun hälytys on aktivoituna.

Harjoitus 2:

Rakenna ohjelmaan ns. snooze-toimito, mikä tarkoittaa sitä että käyttäjä voi siirtää hälytystä eteenpäin esim. 5 minuutilla. Snooze-toiminto pitää tapahtua silloin kun kosketusnäyttöä kosketetaan kellon hälyttäessä. Snooze-toiminnossa kellon hälytys loppuu kosketusnäytön kosketuksen jälkeen ja uusi hälytys tapahtuu esim. tuon 5 minuutin kuluttua. Tässä on huolehdittava että normaali herätysajan asetustoiminto ei käynnisty silloin kun snooze-toiminto on tarkoitus tehdä.

Harjoitus 3:

Vaikka tässä kellossa on jo hieno tapa herätysajan (siis hälytysajan) asettamiseen, lisää siihen kuitenkin painonappi jolla herätysaika voidaan vaihtoehtoisesti asettaa klassiseen Android-tyyliin. Tämä tarkoittaa että herätysaika asetetaan TimePickerDialogin avulla.

Esimerkkiohjelmista löytyy sovellus nimeltä TimePickerDemo, jota tutkimalla saat selville kuinka TimePickerDialogia käytetään. Tuossa esimerkkiohjelmassa käytetään TimePickerFragment-luokkaa joka on sisäinen luokka TimePickerDemoActivity-luokan sisällä. Voit samaan tapaan rakentaan sisäisen luokan AlarmClockActivity-luokan sisään.

AlarmClockView-luokkaan täytyy tässä lisätä metodi jolla hälytystunti ja hälytysminuutti asetetaan.

Kannattaa huomioida myös että TimePickerDemo-ohjelmassa esitellyt asiat eivät toimi sellaisenaan vanhoissa Android-versioissa.

Harjoitus 4:

Paranna ohjelman käytettävyyttä siten että asetettu hälytysaika näytetään isommalla fontilla. Lisää myös ominaisuus että näytöllä näkyy myös aika joka on jäljellä hälytyksen tapahtumiseen. Mm. TimePickerDemo-ohjelmassa on isohko fontti käytössä.

HARJOITUKSIA OHJELMALLA BouncingBall

Huomaa että nämä harjoitukset koskevat nimenomaan sovellusta nimeltä BouncingBall, eivät esim. sovellusta nimeltä BouncingBallsAnimation.

Kyseessä oleva ohjelma on sellainen että se näyttää näytön seinämistä kimpoilevaa palloa joka samalla pyörii. Pallo voidaan ’räjäyttää’ koskettamalla sitä hiirellä (tai sormella). Jos pallo on jo räjähtänyt ja näyttöä kosketetaan, luodaan uusi pallo joka lähtee liikkeelle johonkin satunnaisesti valittuun suuntaan.

Kyseisen ohjelman saa toimimaan kun kopioi projektiin tiedoston

BouncingBallActivity.java

Ohjelman saa toimimaan automaattisesti syntyvällä layout-tiedostolla. Projektia luotaessa on package-nimeksi laitettava `bouncing.ball`

Harjoitus 1:

Muuta ohjelma sellaiseksi että pallon suuntaa voidaan muuttaa näyttöä pyyhkäisemällä. Pyyhkäisyn ei tarvitse, ainakaan aluksi, osua palloon.

Tässä kannattaa katsoa mallia ohjelmasta GesturesDemo, jossa esitetään kuinka Android-sovellus voi reagoida erilaisiin kosketusnäytöllä tehtäviin eleisiin. Näytön pyyhkäisyyn reagoiminen tarkoittaa käytännössä `onFling()`-metodin tekemistä. Tässä on

- laitettava ohjelma toteuttamaan `GestureDetector.OnGestureListener` -rajapinta
- luotava `GestureDetector`-olio
- laitettava `onTouchEvent()`-metodiin `MotionEvent`-tapahtumien siirto `GestureDetector`-oliolle
- tehtävä `onFling()`-metodiin pallon suunnan laskenta
- lisättävä `Bouncer`-luokkaan esim. `set_direction()`-metodi jolla voidaan asettaa pallolle uusi liikkumissuunta

Tätä harjoitusta tehdessä kannattaa ensin varmistaa että pystyt reagoimaan ’flingauksiin’ ja vasta sitten laittaa pallolle uusi suunta. Testausmielessä `onFling()`-metodissa voi aluksi kutsua pallo-oliolle `enlarge()`-metodia, joka suurentaa palloa flingauksen tapahduttua.

`onFling()`-metodille tulee parametreina `MotionEvent`-oliot, jotka sisältävät flingauksen alku- ja loppupisteiden koordinaatit. Näistä olioista ’kaivettavilla’ tiedoilla voi laskea kuinka paljon flingattiin x- ja y-suunnissa. Kun nämä tiedot ovat olemassa, voi käyttää `Math.atan2()`-metodia pallon uuden suunnan laskemiseen.

Seuraavalla sivulla on valmiina `onFling()`-metodin koodi jolla pallolle uusi suunta syntyy.

```
public boolean onFling( MotionEvent first_down_motion,
                        MotionEvent last_move_motion,
                        float velocity_x, float velocity_y )
{
   float movement_x = last_move_motion.getX() - first_down_motion.getX() ;
   float movement_y = last_move_motion.getY() - first_down_motion.getY() ;

   /*  The following code seems to change the direction correctly.
       We must negate the value of the y coordinate so that the
       coordinates correspond to the mathematical system used by atan2().
   */

   movement_y = -movement_y ;

   float  new_direction = (float) Math.atan2( movement_y, movement_x ) ;

   ball_on_screen.set_direction( new_direction ) ;

   return true ;
}
```

`Bouncer`-luokassa pitää olla seuraava metodi jotta yllä annettu ohjelmakoodi toimii:

```
public void set_direction( float new_direction )
{
   bouncer_direction = new_direction ;
}
```

Harjoitus 2:

Muuta ohjelma sellaiseksi että se laskee pallon seinään tapahtuvat törmäykset. Törmäykset havaitaan `Bouncer`-luokan `move()`-metodissa, ja ne pitäisi tulostaa esim. `BouncingBallView`-luokassa. Jotta törmäysmäärätieto saadaan siirtymään oliosta toiseen, täytyy käyttää jotain kikkailua. Eräs tapa on muuttaa `Bouncer`-luokkaa niin että sen olio ’tietää’ monestiko se on törmännyt seinään. Tällöin `Bouncer`-luokkaan tarvitaan datajäsen johon törmäysten määrä talletetaan ja lisäksi ainakin metodi jolla ko. datajäsenen arvo luetaan.

Toinen tapa on käyttää yhtä oliota johon monet oliot pääsevät käsiksi. Kansiosta http://www.naturalprogramming.com/javabookprograms/javafilesextra/ löytyy ohjelma nimeltä **SingletonClassDemo.java** jossa on luokka `CollisionCounter`. Tuo luokka on sellainen että siitä voidaan tehdä vain yksi instanssi eli olio `get_instance()`-metodin avulla, ja monet oliot pääsevät käyttämään tuota yhtä `CollisionCounter`-oliota. Voit kopioida tuon `CollisionCounter`-luokan ohjelmaasi ja sen avulla voit sitten pitää yllä tietoa törmäyksistä.

Törmäysten määrän voi tulostaa esim. näytön alareunaan.

Harjoitus 3:

Tee ohjelmasta ’täydellinen peli’ siten että kun pallo on törmännyt riittävän monta kertaa seiniin, niin se räjähtää, ja uusi peli on mahdollista aloittaa.

Androidin valmiilla luokilla voi rakentaa erilaisia dialogeja, joilla voi kysyä esim. että haluaako käyttäjä uuden pelin. Opettajan Android-esimerkkiohjelmien joukosta löytyy sovellus nimeltä AlertDialogDemo, jossa on esitetty ainakin yksi tapa jolla saadaan näkymään dialogi jolla uusi peli voitaisiin käynnistää.