

(SK-VM)

(가 :)

()

가 가

가

SKVM

가

SKVM

가

SKVM

가

가

..^^

가

가

가..ㅎㅎㅎ Timer가

Timer가

..^^

TorusCanvas.java

```
import java.io.*;
import java.util.*;
import javax.microedition.lcdui.*;

import com.xce.lcdui.*;
```

```

import com.xce.io.*;
import com.skt.m.*;

public class TorusCanvas extends Canvas implements XTimerListener
{
    /*
    ** -----
    **      define
    ** -----
    */

    //      LCD      (120-144      .)
    public static int LCD_WIDTH      = 120;
    public static int LCD_HEIGHT     = 144;
    public static int LCD_X          = (XDisplay.width - LCD_WIDTH)
/ 2;
    public static int LCD_Y          =      (XDisplay.height2 -
LCD_HEIGHT) / 2;

    //
    public final static int TORUS_INTRO  = 0;      //
    public final static int TORUS_READY  = 1;      //
    public final static int TORUS_GO     = 2;      //      .
    public final static int TORUS_OVER   = 3;      //

    // Bar : 가 ,      Screw :      .
    public final static int[][] TORUS_STAGE = {{3, 7, 3},
        {4, 8, 4}, {5, 9, 5}, {6, 10, 6}, {7, 11, 7},
        {8, 12, 8}, {9, 13, 9}, {10, 14, 10}, {11, 15, 11}, {12, 16, 12}};

    public final static int MAX_COLOR    = 4;      // Torus      .

```

```

        public final static int LEVEL_UP_CONDITION = 300;        //        .(
                                .)

        // Torus가
        public final static int MOTION_Y = LCD_Y + 12;

/*
** -----
**      Class
** -----
*/

        private int REAREA_X;                                //        X
        private int REAREA_Y;                                //        Y
        private int REAREA_WIDTH;        //        WIDTH
        private int REAREA_HEIGHT;        //        HEIGHT

        TorusMIDlet        tMIDlet;        // MIDlet
        TorusEngine        tEngine;        // Engine
        XTimer        xTimer;        // Timer

/*
** -----
**
** -----
*/

        int aniIntro;                                // Intro        .
        int statusTorus = TORUS_READY;        //        .
        int levelUp;                                //

        Graphics2D g2 = Graphics2D.getGraphics2D(Toolkit.graphics);

/*

```

```

** -----
**      Image
** -----
*/

    Image imgIntro;          //
    Image imgSlot;           // SLOT
    Image imgTorus;          // .
    Image imgScrewBar;       // Screw Bar
    Image imgScrewPin;       // Screw Pin
    Image[] imgBG = new Image[3]; //

/*
** -----
**      Method
** -----
*/

/*
** -----
**      Function:
**          - TorusCanvas
**
**      Description:
**          -
**
**      Input:
**
**      Return value:
**
** -----
*/

```

```

TorusCanvas(TorusMIDlet tMIDlet)
{
    this.tMIDlet = tMIDlet;
    tEngine = new TorusEngine(this);

    try
    {
        //
        //      PNG
        // LBM      Mixer      ..
        //      가
        //      가      ..
        imgIntro      = Image.createImage("/img/Intro.png");
//
        imgSlot      = Image.createImage("/img/Slot.png");
//
        imgTorus      = Image.createImage("/img/Torus.png");
//
        imgScrewBar      = Image.createImage("/img/ScrewBar.png");
//
        imgScrewPin      = Image.createImage("/img/ScrewPin.png");
//
        imgBG[0]      = Image.createImage("/img/BG1.png");
// Back Ground
        imgBG[1]      = Image.createImage("/img/BG2.png");
// Back Ground
        imgBG[2]      = Image.createImage("/img/BG3.png");
// Back Ground
    } catch(Exception e){}

    //
    tEngine.startInit(0);

    //
    //      300
    xTimer = new XTimer(300);

```

```

        xTimer.setTimerListener(this);

        setRefreshArea(0, 0, XDisplay.width, XDisplay.height2);
        xTimer.start();
    }

```

```

/*
** -----
** Function:
**         setRefreshArea
**
** Description:
**         repaint          .
**
** Input:
**         x :
**         y :
**         width :
**         height :
**
** Return value:
**
** -----
*/

    public void setRefreshArea(int x, int y, int width, int height)
    {
        REAREA_X = x;
        REAREA_Y = y;
        REAREA_WIDTH = width;
        REAREA_HEIGHT = height;
    }

    public void repaintArea()

```

```

    {
        repaint(REAREA_X, REAREA_Y, REAREA_WIDTH, REAREA_HEIGHT);
    }

    // serviceRepaints();
}

/**
 * -----
 * ** Function:
 * ** performed
 * **
 * ** Description:
 * **
 * **
 * ** Input:
 * **
 * ** Return value:
 * **
 * -----
 */

public void performed(XTimer aXTimer)
{
    switch(levelUp++)
    {
        // Screw가
        case LEVEL_UP_CONDITION:
            // Back Color
            Toolkit.graphics.setColor(0x000000);
            Toolkit.graphics.fillRect(tEngine.SLOT_X,
tEngine.SCREWPIN_Y,
tEngine.SLOT_WIDTH,
tEngine.SCREWPIN_HEIGHT);
            TorusEngine.SCREWBAR_HEIGHT);
    }
}

```

```

        // Screw
        tEngine.screwLevelUp();

        // Screw
        tEngine.drawScrewTorus(Toolkit.graphics);
        break;

//
case LEVEL_UP_CONDITION * 2:
    //
    //
    levelUp = 0;

    // 가
    tEngine.slotLevelUp();

    //
    statusTorus = TORUS_READY;

    setRefreshArea(0, 0, XDisplay.width, XDisplay.height2);
    break;
}

repaint(REAREA_X, REAREA_Y, REAREA_WIDTH, REAREA_HEIGHT);
serviceRepaints();
}

```

/*

** -----

** Function:

** paint

**

** Description:

**


```

** Input:
**
** Return value:
**
** -----
*/

    public void paint(Graphics g)
    {
        switch(statusTorus)
        {
            case TORUS_INTRO:
                break;

            case TORUS_READY:
                //
                g.drawImage(imgBG[aniIntro++ % 3], LCD_X + LCD_WIDTH /
2, LCD_Y + LCD_HEIGHT / 2, g.HCENTER | g.VCENTER);

                // Torus가
                g.setColor(0x000000);
                g.fillRect(tEngine.SLOT_X, MOTION_Y,
                    tEngine.SLOT_WIDTH, tEngine.SLOT_Y -
MOTION_Y);

                // Slot
                tEngine.drawSlotTorus(g);

                // Screw
                tEngine.drawScrewTorus(g);

                //
                statusTorus = TORUS_GO;

                // Repaint
                // Torus가 Repaint
                setRefreshArea(tEngine.SLOT_X, 0,

```

```

                                tEngine.SLOT_WIDTH, LCD_Y + LCD_HEIGHT);

                                break;

                                case TORUS_GO:
                                    tEngine.paint(g);
                                    break;

                                case TORUS_OVER:
                                    break;

                                }

        }

/*
** -----
** Function:
**         keyPressed
**
** Description:
**         -
**
** Input:
**         // SK-VM Specific
**         KEY_CLR          = 8;
**
**         KEY_POUND        = 35;
**         KEY_NUM0         = 48;
**         KEY_NUM1         = 49;
**         KEY_NUM2         = 50;
**         KEY_NUM3         = 51;
**         KEY_NUM4         = 52;
**         KEY_NUM5         = 53;
**         KEY_NUM6         = 54;
**         KEY_NUM7         = 55;
**         KEY_NUM8         = 56;
**         KEY_NUM9         = 57;

```

```

**          KEY_STAR          = 42;
**
**          KEY_COML          = 129;
**          KEY_COMC          = 130;    // RESERVED
**          KEY_COMR          = 131;
**
**          KEY_UP            = 141;
**          KEY_LEFT          = 142;
**          KEY_RIGHT         = 145;
**          KEY_DOWN          = 146;
**          KEY_FIRE          = 148;
**
**          KEY_CALL          = 190;
**          KEY_END           = 191;
**
**          KEY_FLIP_OPEN     = 192;
**          KEY_FLIP_CLOSE    = 193;
**          KEY_VOL_UP        = 194;
**          KEY_VOL_DOWN      = 195;
**
**  Return value:
**
**  -----
**/

    public void keyPressed(int keyCode)
    {
        switch(statusTorus)
        {
            case TORUS_INTRO:
                break;

            case TORUS_GO:
                tEngine.keyPressed(keyCode);
                break;

```

```

        case TORUS_OVER:
            break;
    }
}

/*
** -----
**  Function:
**      keyReleased
**
**  Description:
**
**
**  Input:
**
**  Return value:
** -----
**/

    public void keyReleased(int keyCode)
    {
/*      switch(statusTorus)
        {
            case TORUS_INTRO:
                break;

            case TORUS_GO:
                tEngine.keyReleased(keyCode);
                break;

            case TORUS_OVER:
                break;
        }
*/
    }
*/

```

```
}  
  
}
```

TorusEngine.java

```
import java.io.*;  
import java.util.*;  
import javax.microedition.lcdui.*;  
  
import com.xce.lcdui.*;  
import com.xce.io.*;  
import com.skt.m.*;  
  
public class TorusEngine  
{  
  
    /*  
    ** -----  
    **      define  
    ** -----  
    */  
  
    //  
    // SLOT  
    public final static int SLOT_UBAR_WIDTH      = 12;    // SLOT      Torus가      가  
  
    public final static int SLOT_UBAR_HEIGHT = 12;        // SLOT      Torus가      가  
    .  
  
    public int SLOT_WIDTH;          //  
    public int SLOT_HEIGHT;         //  
    public int SLOT_X;              //      X  
    public int SLOT_Y;              //      Y  
  
    // Screw  
    // Bar  
    public final static int SCREWBAR_WIDTH      = 11;    //
```

```
public final static int SCREWBAR_HEIGHT = 4; //
public int SCREWBAR_X ; //
X
public int SCREWBAR_Y; //
Y

// Pin
public final static int SCREWPIN_WIDTH = 11; //
public int SCREWPIN_HEIGHT; //

public int SCREWPIN_X ; // X
public int SCREWPIN_Y; // Y

// Torus
public final static int TORUS_WIDTH = 11; // Torus
public final static int TORUS_HEIGHT = 11; // Torus
public final static int TORUS_SLOT_HEIGHT = 3; // Torus

public final static int TORUS_NOMOTION = 26; // Torus
Y
public final static int TORUS_MOTION = 4; // 4가
가

/*
** -----
** Class
** -----
*/

TorusCanvas tCanvas;

// Torus
// 가
// 가 가
```

```

        //
        TorusRing[] tRing = new TorusRing[16];

/*
** -----
**
** -----
*/

// SLOT
//
//
// .. Torus가
int[][] TORUS_SLOT = new int[12][16]; // SLOT 2
int[] slotCurLine = new int[16]; // SLOT
.
int[] slotDelLine = new int[16]; //
int slotMaxLine; // SLOT 가
.
int slotCount; // SLOT

// Screw
int[] TORUS_SCREW = new int[12]; // .( )
int screwCurStatck; //
int maxOfscrewStack; // Screw Torus
.
int screwPos; //

int curLevel; // ..

boolean bDelLine; //
.
/*

```

```

** -----
** Function:
**         TorusEngine
**
** Description:
**
**
** Input:
**
** Return value:
**
** -----
*/

    TorusEngine(TorusCanvas tCanvas)
    {
        this.tCanvas = tCanvas;
    }

/*
** -----
** Function:
**         initStage
**
** Description:
**
**
** Input:
**
** Return value:
**
** -----
*/

```



```

public void statgelnit(int level)
{
    // LEVEL
    curLevel = level;

    /////////// SLOT
    // SLOT
    slotCount = TorusCanvas.TORUS_STAGE[curLevel][0];

    // SLOT Torus
    slotMaxLine = TorusCanvas.TORUS_STAGE[curLevel][1];

    // Screw Torus
    maxOfscrewStack = TorusCanvas.TORUS_STAGE[curLevel][2];

    // SLOT
    // SLOT 가 X
    SLOT_WIDTH = SLOT_UBAR_WIDTH * slotCount + 1;
    SLOT_X = TorusCanvas.LCD_X + (TorusCanvas.LCD_WIDTH - SLOT_WIDTH) / 2;

    /////////// Screw
    // Screw ⊥ ..
    // ⊥ = | (Pin) + — (Bar) 가
    // Screw Bar X, Y
    screwCurStatck = 0;

    SCREWBAR_Y = TorusCanvas.LCD_Y + TorusCanvas.LCD_HEIGHT - 5 -
SCREWBAR_HEIGHT;
    SCREWBAR_X = SLOT_X + 1;

    // Screw Pin
    // Screw X, Y
    //
    SCREWPIN_HEIGHT = maxOfscrewStack * TORUS_SLOT_HEIGHT;
    SCREWPIN_Y = SCREWBAR_Y - SCREWPIN_HEIGHT;

```

```
        SCREWPIN_X = SCREWBAR_X;

        screwCurStatck = 0;

        // SLOT
        SLOT_HEIGHT = SLOT_UBAR_HEIGHT + slotMaxLine * TORUS_SLOT_HEIGHT;//
SLOT      가 Torus      3      .
        SLOT_Y = SCREWPIN_Y - SLOT_HEIGHT;

        //      Torus      .
        for(int i = 0; i < slotCount; i++)
        {
            if(tRing[i] == null)
                tRing[i] = new TorusRing(tCanvas);

            tRing[i].setStaus(i);

            //      .

            slotCurLine[i] = 0;
        }
    }

/*
** -----
** Function:
**      - screwLevelUp
**
** Description:
**      -      가      .(Torus      가      .)
**
** Input:
**
** Return value:
**
** -----
```

```

*/

    public void screwLevelUp()
    {
        // Screw Torus 가 .
        maxOfscrewStack - - ;

        // Screw Bar .
        SCREWBAR_Y -= TORUS_SLOT_HEIGHT;

        // Screw Pin .
        SCREWPIN_HEIGHT -= TORUS_SLOT_HEIGHT;

        // Screw Torus가 Torus .
        if(screwCurStatck >= maxOfscrewStack)
            screwCurStatck - - ;
    }

/*
** -----
** Function:
**         - slotLevelUp
**
** Description:
**         - .
**
** Input:
**
** Return value:
**
** -----
*/

    public void slotLevelUp()
    {
        statgeInit(++curLevel);
    }

```

```

/*
** -----
** Function:
**         - slotPut
**
** Description:
**         - Slot      Put
**
** Input:
**         - curSlot :
**         - torusColor :
**
** Return value:
**         - ,
**
** -----
*/

    public boolean slotPut(int curSlot, int torusColor)
    {
        //
        // if(slotCurLine[curSlot] >= slotMaxLine || curSlot >= slotCurLine.length ||
        //     torusColor > TorusCanvas.MAX_COLOR)
        //     return false;

        //
        // if(slotCurLine[curSlot] >= slotMaxLine)
        //     return false;

        // SLOT
        TORUS_SLOT[curSlot][slotCurLine[curSlot]++] = torusColor;

        return true;
    }

```

```

/*
** -----
** Function:
**         - slotGet
**
** Description:
**         - Slot      Get
**
** Input:
**         - curSlot :
**
** Return value:
**         - 0 :      ..      ..
**         - 1 Over : Color Value
**
** -----
*/

    public int slotGet(int curSlot)
    {
        //
        // if(TORUS_SLOT[curSlot][0] == 0 || curSlot >= slotCurLine.length)
        //         return 0;

        //
        // if(TORUS_SLOT[curSlot][0] == 0)
        //         return 0;

        //         Torus
        int colorValue = TORUS_SLOT[curSlot][0];

        //
        //
        System.arraycopy(TORUS_SLOT[curSlot], 1, TORUS_SLOT[curSlot], 0,
slotCurLine[curSlot] - -);

```

```

        return colorValue;
    }

/*
** -----
** Function:
**         - slotPush
**
** Description:
**         - Slot          Push
**
** Input:
**         - curSlot :
**         - torusColor :
**
** Return value:
**         - ,
**
** -----
*/

    public boolean slotPush(int curSlot, int torusColor)
    {
        //
        // if(slotCurLine[curSlot] >= slotMaxLine || curSlot >= slotCurLine.length ||
        //         torusColor > TorusCanvas.MAX_COLOR)
        //         return false;

        //
        if(slotCurLine[curSlot] >= slotMaxLine)
            return false;

        for(int i = slotCurLine[curSlot]++; i >= 1; i--)
            TORUS_SLOT[curSlot][i] = TORUS_SLOT[curSlot][i - 1];

```

```

        TORUS_SLOT[curSlot][0] = torusColor;

        return true;
    }

/*
** -----
** Function:
**         - screwPush
**
** Description:
**         - Screw      Push
**
** Input:
**         - curSlot :
**
** Return value:
**
** -----
*/

    public void screwPush(int curSlot)
    {
        if(screwCurStatck >= maxOfscrewStack)
            return;

        // SLOT      Get
        int popColor = slotGet(curSlot);

        if(popColor > 0)
            TORUS_SCREW[screwCurStatck++] = popColor;
    }

```

```

/*
** -----
** Function:
**         - screwPop
**
** Description:
**         - Screw      Pop
**
** Input:
**         - curSlot :
**
** Return value:
**
** -----
*/

```

```

public void screwPop(int curSlot)
{
    if(screwCurStatck <= 0)
        return;

    // SLOT      Push
    if(slotPush(curSlot, TORUS_SCREW[screwCurStatck - 1]))
        TORUS_SCREW[ - -screwCurStatck] = 0;
}

```

```

/*
** -----
-----
** Function:
**         deleteLine
**
** Description:
**         -      .(      가      가      .)
**

```



```

** Input:
**
** Return value:
**
** -----
**
*/

public synchronized boolean delLine()
{
    int i = 0, j = 0, k = 0;
    boolean bDel = false;

    int torusColor = 0;
    int countMinSlot = 255;

    for(i = 0; i < slotCount; i++)
    {
        //
        if(slotCurLine[0] < 1)
            return bDel;
        else
        {
            // 가
            if(slotCurLine[i] < countMinSlot)
                countMinSlot = slotCurLine[i];
        }
    }

    // 가
    for(i = 0, k = 0; i < countMinSlot; i++)
    {
        //
        torusColor = TORUS_SLOT[0][i];
        bDel = true;
        slotDelLine[k] = -1;
    }
}

```

```

        for(j = 0; j < slotCount; j++)
        {
            //
            if(torusColor != TORUS_SLOT[j][i])
            {
                bDel = false;
                break;
            }
        }

        //
        if(bDel)
            slotDelLine[k++] = i;
    }

    //
    if(k > 0)
    {
        //
        for(i = 0; i < k; i++)
        {
            for(j = 0; j < slotCount; slotCurLine[j] --, j++)
                System.arraycopy(TORUS_SLOT[j], slotDelLine[i] +
1, TORUS_SLOT[j], slotDelLine[i], slotCurLine[j] - slotDelLine[i]);
        }

        bDel = true;
    }else
        bDel = false;

    return bDel;
}

```

```

/*
** -----
** Function:
**         drawSlotTorus
**
** Description:
**         Slot, Torus
**
** Input:
**
** Return value:
** -----
*/

public void drawSlotTorus(Graphics g)
{
    int i = 0, j = 0;

    // Torus가
    int torusWidth = TORUS_WIDTH + 1;
    int torusHeight = 3;
    int torusX = SLOT_X + 1;
    int torusY = SLOT_Y + SLOT_HEIGHT - torusHeight;

    // drawSlot
    //         가         가         for
    //
    tCanvas.g2.drawImage(SLOT_X, SLOT_Y,
                        tCanvas.imgSlot, 0, 0,
                        SLOT_WIDTH, SLOT_HEIGHT, Graphics2D.DRAW_COPY);

    // drawTorus
    //         Slot         (2         )         가         Torus
    for(i = 0; i < slotCount; i++)

```

```

        {
            //          Torus          .
            for(j = 0; j < slotCurLine[i]; j++)
            {
                tCanvas.g2.drawImage(torusX + torusWidth * i,
                                     torusY - torusHeight * j,
                                     tCanvas.imgTorus,
                                     (TORUS_SLOT[i][j] - 1) * TORUS_WIDTH,
TORUS_NOMOTION,
                                     TORUS_WIDTH,
                                     torusHeight,
Graphics2D.DRAW_COPY);
            }
        }
    }

/*
** -----
** Function:
**         drawScrewTorus
**
** Description:
**         Screw, Torus          .
**
** Input:
**
** Return value:
**
** -----
*/

    public void drawScrewTorus(Graphics g)
    {
        int screwWidth = SCREWBAR_WIDTH + 1;

        // Back Color

```

```

        // ScrewPin
        g.setColor(0x000000);
        g.fillRect(SLOT_X, SCREWPIN_Y, SLOT_WIDTH, SCREWPIN_HEIGHT +
SCREWBAR_HEIGHT);

        // drawBarScrew
        g.drawImage(tCanvas.imgScrewBar,
                    SCREWBAR_X + screwPos * screwWidth, SCREWBAR_Y, g.LEFT |
g.TOP);

        // drawPinScrew
        tCanvas.g2.drawImage(SCREWPIN_X + screwPos * screwWidth, SCREWPIN_Y,
                             tCanvas.imgScrewPin, 0, 0,
                             SCREWPIN_WIDTH, SCREWPIN_HEIGHT, Graphics2D.DRAW_COPY);

        // drawTorus
        int torusX = SCREWBAR_X + screwPos * screwWidth;
        int torusY = SCREWPIN_Y + SCREWPIN_HEIGHT - TORUS_SLOT_HEIGHT;

        // Screw Torus
        for(int i = 0; i < screwCurStatck; i++)
        {
            tCanvas.g2.drawImage(torusX, torusY - TORUS_SLOT_HEIGHT * i,
                                tCanvas.imgTorus,
                                (TORUS_SCREW[i] - 1) * TORUS_WIDTH,
TORUS_NOMOTION,
                                TORUS_WIDTH,
                                TORUS_SLOT_HEIGHT,
                                Graphics2D.DRAW_COPY);
        }
    }

/*
** -----

```

```

** Function:
**         paint
**
** Description:
**
**
** Input:
**
** Return value:
**
** -----
*/

public void paint(Graphics g)
{
    //
    if(bDelLine)
    {
        //      가      Slot      .
        if(delLine())
            drawSlotTorus(g);

        bDelLine = false;
    }

    //      Torus      .
    for(int i = 0; i < slotCount; i++)
    {
        switch(tRing[i].paint(g))
        {
            //      .
            case -1:
                return;

            // Torus가
            //      ..      Torus가      .

```

```

                                case 1:
                                    bDelLine = true;

                                    // Torus가
                                    // Slot
                                    drawSlotTorus(g);
                                    break;
                                }
                            }
                        }

/*
** -----
** Function:
**         keyPressed
**
** Description:
**
**
** Input:
**
** Return value:
** -----
*/

public void keyPressed(int keyCode)
{
    //         가         XDisplay.refresh()
    switch(keyCode)
    {
        // Screw         SLOT
        case Canvas.KEY_UP://         = 141;
            screwPop(screwPos);

            // Torus
            //

```

```

        bDelLine = true;

        //
        drawScrewTorus(Toolkit.graphics);
        drawSlotTorus(Toolkit.graphics);
        XDisplay.refresh(SLOT_X,      SLOT_Y,      SLOT_WIDTH,
SLOT_HEIGHT + SCREWPIN_HEIGHT + SCREWBAR_HEIGHT);
        break;

// Screw
//
case Canvas.KEY_LEFT:// = 142;
    screwPos = (--screwPos < 0)?0:screwPos;

//
    drawScrewTorus(Toolkit.graphics);
    XDisplay.refresh(SCREWPIN_X, SCREWPIN_Y, SLOT_WIDTH,
SCREWPIN_HEIGHT + SCREWBAR_HEIGHT);
    break;

// Screw
//
case Canvas.KEY_RIGHT:// = 145;
    screwPos = (++screwPos >= slotCount)?
        slotCount - 1:screwPos;

//
    drawScrewTorus(Toolkit.graphics);
    XDisplay.refresh(SCREWPIN_X, SCREWPIN_Y, SLOT_WIDTH,
SCREWPIN_HEIGHT + SCREWBAR_HEIGHT);
    break;

// SLOT      Screw
case Canvas.KEY_DOWN:// = 146;
    screwPush(screwPos);

```



```

        // Torus
        //
        bDelLine = true;

        //
        drawScrewTorus(Toolkit.graphics);
        drawSlotTorus(Toolkit.graphics);
        XDisplay.refresh(SLOT_X,      SLOT_Y,      SLOT_WIDTH,
SLOT_HEIGHT + SCREWPIN_HEIGHT + SCREWBAR_HEIGHT);
        break;
    }
}

/*
** -----
**  Function:
**      keyReleased
**
**  Description:
**
**
**  Input:
**
**  Return value:
**
** -----
*/
    public void keyReleased(int keyCode)
    {
    }
}

```

TorusRing.java

```

import java.io.*;
import java.util.*;

```

```

import javax.microedition.lcdui.*;

import com.xce.lcdui.*;
import com.xce.io.*;
import com.skt.m.*;

public class TorusRing
{
    public final static int GAME_LEVEL = 9;

    int x;                // Torus X
    int y;                // Torus Y
    int color;            // Torus .
    int motion;           // Torus
    int selSlot;          // Torus .
    int speed;            // Torus가

    Random random;
    TorusCanvas tCanvas;

    /*
    ** -----
    ** Function:
    **         TorusRing
    **
    ** Description:
    **
    **
    ** Input:
    **
    ** Return value:
    **
    ** -----
    */

```

```

        TorusRing(TorusCanvas tCanvas)
        {
            this.tCanvas = tCanvas;
        }

    /*
    ** -----
    ** Function:
    **         setStaus
    **
    ** Description:
    **         .
    **
    ** Input:
    **
    ** Return value:
    **
    ** -----
    */

    public void setStaus(int selSlot)
    {
        //
        this.selSlot = selSlot;

        //
        motion = 0;

        //
        // ..
        random = new Random(System.currentTimeMillis());
        color = Math.abs(random.nextInt() + selSlot) % TorusCanvas.MAX_COLOR;

        // , 가 .
        speed = ((color + 1) * TorusCanvas.MAX_COLOR) % GAME_LEVEL;

        // X, Y

```

```

        x = tCanvas.tEngine.SLOT_X + 1 + (TorusEngine.TORUS_WIDTH + 1) * selSlot;
        y = TorusCanvas.MOTION_Y;

        if(random != null)
            random = null;
    }

/*
** -----
** Function:
**     paint
**
** Description:
**
**
** Input:
**
** Return value:
** -----
*/

    public int paint(Graphics g)
    {
        //     Torus
        g.setColor(0x000000);
        g.fillRect(x, y, TorusEngine.TORUS_WIDTH, TorusEngine.TORUS_HEIGHT);

        //     가
        y += speed;

        //     Torus
        tCanvas.g2.drawImage(x, y,
            tCanvas.imgTorus,
            TorusEngine.TORUS_WIDTH * color,

```

```
TorusEngine.TORUS_HEIGHT * (motion++ %
TorusEngine.TORUS_MOTION),
TorusEngine.TORUS_WIDTH, TorusEngine.TORUS_HEIGHT,
Graphics2D.DRAW_COPY);

// Torus가 ..
if(y >= tCanvas.tEngine.SLOT_Y)
{
    // 가 ..
    if(!tCanvas.tEngine.slotPut(selSlot, color + 1))
        return -1;

    //
    setStaus(selSlot);
    return 1;
}

return 0;
}
}
```

Torus 가 가 ..
(, ,), , 가
.

가 . 가 .. Torus
TT ...

.. 가 .. Torus ..^^

:
:
: pinkred@hanafos.com

: ..

