핑크레드의 실전 모바일 자바(SK-VM) 프로그래밍
(가칭 : 구루구루 춥)

안녕하세요 핑크레드 입니다.

벌써 4번째 강좌가 시작 되었네요..별 도움이 안되고 있는듯한 느낌이 들기도 하고요..ㅎㅎㅎ 그래도 언제나 이것을 보고 도움이 되시는 분이 있으시면 감사할 따름이죠.

이번 강좌는 말씀드렸듯이 점수, 인트로, 레벨선택, 게임오버등등의 마무리 처리 부분입니다. 이것들은 게임의 핵심적인 부분은 아니지만 부요소로서 꼭 필요한 부분이죠. 제가 디자이너가 아니라서 부족한 점이 많지만..그냥 어련하겠어라고 넘어가시면 좋을 듯 합니다.

이렇게 소스만 제공하니깐..내용도 부실한 느낌이 들기는 합니다. 이게 머야..이런식이죠. 앞에 몇마디 하다가 뒤에 소스 공개..별도움이 안된다고 생각합니다. 예전 처럼 약간의 팁이나 현재 돌아가는 야그들도 좋을 것 같습니다.

해당 게임을 GNEX로 컨버팅 했습니다. 전체다는 아니고 강좌3정도의 분량만 했습니다. 왜 그랬을까 ㅜㅜ. 이유는 간단합니다. GNEX를 익혀보겠다는 의미이죠. 프로그래머는 음식을 가리면 안된다고 생각합니다. 모든 것을 잘할수도 없고 한가지만이라도 잘하기도 힘든 세상입니다. 하지만 자기의 주종목이 있으면 부종목도 있어야 하지 않을까라는 생각이 듭니다. 언제까지 자바만 하고 있을 수는 없거든요. 여러분들도 자바에만 매달리지 마십시오. 자바란 단순하게 보여주기 위한 도구일뿐입니다. 소비자는 컨텐츠가 무슨 언어로 짜여있는지를 생각하지 않습니다. 단지 좋다. 나쁘다. 등을 잼미있다. 이런식의 느낌을 표현하고 즐길 뿐입니다. 그렇기 때문에 해당 컨텐츠를 만들기 위한 적절한 언어 선택은 플랫폼의 능력이며 해당언어로 선택했을 때 무리 없이 진행되는 플랫폼을 선택하게 되어있습니다. 무조건 자바가 느리다고 사용을 안하는 것도 아니고 무조건 빠르다고 C를 사용하는 것도 아닙니다. 해당 컨텐츠에 맞추어서 플랫폼을 선택하는 것이 적절 하다고 생각됩니다.

제가 C쪽은 GNEX 말고 다른 플랫폼에서 해본적이 있어서 GNEX도 어렵지 않을 것 같아서 해봤는데 GNEX가 제약이 점 있는 것 같습니다. 이런 제약만 잘 지키면 GNEX도 어렵지 않고 오히려 편할수도 있죠. 단지 언제나 전체적이 그림이 중요하게 되네요. GNEX의 무분별한 paint()는 건강에 해롭되는 것이죠. 이런 문법이 아닌 전체적인 통로가 잘 만들어야지 누가 보기에도 아 잘만들었구나라는 말이 나오는 것 같습니다.

왜 갑자기 GNEX냐 하는 말을 듣는다면 전 이렇게 말씀드리고 싶습니다. 현재까지 GVM이 SKT의 게임 시장을 거의 독차지 하고 있었습니다. 이러한 것은 누구나 다 알고 있고 또한

게임 랭크 20안의 순위는 거의 GVM입니다. 이런시점에서 SKVM은 망하느냐라는 의문이 있었을 때 위피가 나오면서 오히려 GVM이 망하게 될 것 처럼 보였습니다. 이러한 시점에 GNEX가 나왔고 이제는 GVM, GNEX, SKVM이 공존하는 때가 되었습니다. 그럼 이것이 무엇을 의미하느냐 이제는 폰의 사양이 좋아져서 소비자의 욕구에 맞는 컨텐츠를 만들기에는 GVM이 받쳐주지를 못합니다. 속도가 아니고 퀄러티면에서 SKVM보다 떨어지게 되죠. 그렇다면 GNEX로 만들면 되지 않겠느냐라고 할 때의 문제는 아직 단말 공급이 부족하다는 것이지요. 이러한 시점이 바로 지금입니다. 그래서 결국은 양쪽을 다 선점할수 있는 SKVM으로 현시점에서는 만들수 밖에 없다는 것이죠. 그렇지만 향후 GNEX 단말이 퍼지는 순간에는 다시 GNEX를 넘어갈 것 이라고 생각됩니다. 이런시점에서 SKVM은 지금을 위한 플랫폼이고 GNEX는 앞으로를 위한 플랫폼이라고생각됩니다. 그래서 조금씩 공부를 해두는 것도 그리 나쁘지 않을 것이라고 생각됩니다. 그리고 부분별한 C코드 보다는 형태가 잡혀있는 자바를 공부하고 나서 이것을 C에 적용했을 때 더 멋진 형태가 나오는 것 같습니다. C의 형태가 나쁘다고 말씀드리는 것은 아닙니다. C로 시작했을 경우와 자바로 시작했을 경우 프로그램 형태를 즉 전체적인 왁구를 보게 되면서 느끼게 될것입니다. 여기서 딴지 그럼 위피는 모냐라고 묻는다면 그냥 한귀로 듣고 한귀로 흘릴렵니다. 그냥 위피는 위피 일뿐이라고 말씀드리고 싶습니다. 위피에 대해서는 특별히 말씀드릴 것이 없습니다. 아시는 분은 다 아시리라 생각됩니다. 모르시면 그냥 모르시는 데로 넘어가시면 될 것 같습니다.

그럼 소스를 공개 하도록 하겠습니다. 아직도 수정할수 있는 부분이 많다고 생각됩니다. 제가 무슨 IQ200도 아니고 하기 때문에 더 좋은 방법은 수도 없이 많다고 생각됩니다. 그러므로 해보시면서 이게 아닌 것 같은데라고 생각되시면 고쳐서 사용해 보십시오. 그래야지 실력이 늡니다. 전체적이 설명은 언제나 주석을 많이 달았습니다. 참고 하시고 프로그래밍 하시면 될 듯 합니다. 전체 소스는 다음 강좌때 배포 하도록 하겠습니다.

TorusMIDlet.java

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

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

public class TorusMIDlet extends MIDlet
{
```

```
/*
** ----------------------------------------------------------------
**        변수 선언
** ----------------------------------------------------------------
*/
        public final static String TORUS_FILE_INFO      = "/torusInfo.txt";

        Display display;                                // Display 변수
        TorusCanvas tCanvas;                            // Canvas 변수



/*
** ----------------------------------------------------------------
**  Function:
**          startApp
**
**  Description:
**          어플리케이션이 시작되면 실행됨
**
**  Input:
**
**  Return value:
**
** ----------------------------------------------------------------
*/

        public void startApp()
        {
                Device.setBacklightEnabled(false);
                BackLight.on(0);

                display = Display.getDisplay(this);
                tCanvas = new TorusCanvas(this);
                display.setCurrent(tCanvas);
        }
```

```
/*
** ---------------------------------------------------------------
**  Function:
**          pauseApp
**
**  Description:
**          어플리케이션이 멈출경우 호출됨(잘 동작하지 않음)
**
**  Input:
**
**  Return value:
**
** ---------------------------------------------------------------
*/
        public void pauseApp() {}

/*
** ---------------------------------------------------------------
**  Function:
**          resumeApp
**
**  Description:
**          어플리케이션 pauseApp() 후에 복귀 할때 호출됨(실제로 동작하지 않음)
**
**  Input:
**
**  Return value:
**
** ---------------------------------------------------------------
*/

        public void resumeApp(){}

/*
```

```
** ----------------------------------------------------------------
**  Function:
**          destroyApp
**
**  Description:
**          어플리케이션 종료 될때 호출됨
**
**  Input:
**
**  Return value:
**
** ----------------------------------------------------------------
*/
        public void destroyApp(boolean con){}

/*
** ----------------------------------------------------------------
**  Function:
**          readInfo
**
**  Description:
**              - 최고 점수를 읽어온다.
**
**  Input:
**
**  Return value:
**
** ----------------------------------------------------------------
*/

        public String readInfo()
        {
                String info = null;

                try
                {
```

```
                        if(XFile.exists(TORUS_FILE_INFO))
                        {
                                XFile xFile = new XFile(TORUS_FILE_INFO, XFile.READ);

                                byte[] lenBuf = new byte[xFile.available()];

                                xFile.read(lenBuf, 0, lenBuf.length);
                                xFile.close();
                                xFile = null;

                                info = new String(lenBuf);
                        }
                }catch(Exception e){}

                return info;
        }


/*
** ----------------------------------------------------------------
**  Function:
**          writeInfo
**
**  Description:
**              - 최고 점수를 저장한다.
**
**  Input:
**
**  Return value:
**
** ----------------------------------------------------------------
*/

        public void writeInfo(String name, int score)
        {
                if(name == null)
```

```
                    name = "핑크레드";

                try
                {
                    XFile xFile = null;

                    byte[] lenBuf = (name + ":" + score).getBytes();

                    xFile = new XFile(TORUS_FILE_INFO, XFile.WRITE);
                    xFile.write(lenBuf, 0, lenBuf.length);
                    xFile.close();
                    xFile = null;
                    lenBuf = null;
                }catch(Exception e){}
        }


}
```

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 선언
** ----------------------------------------------------------------
*/
```

```
        // 단말 LCD 정보(120-144 단말을 기준으로 한다.)
        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;



        // 상태
        public final static int TORUS_INTRO   = 0;              // 인트로
        public final static int TORUS_READY   = 1;              // 시작
        public final static int TORUS_GO              = 2;              // 게임 중.
        public final static int TORUS_OVER    = 3;              // 게임 오버


        // Bar : 가로, 세로   Screw : 세로.
        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 최대 색상 개수.
        public final static int LEVEL_UP_CONDITION = 300;       //레벨업 조건.

        // Torus가 떨어지는 높이
        public final static int MOTION_Y = LCD_Y + 12;
        public final static int MAX_STRING = 20;                        // 최대 이름 입력.


/*
** ----------------------------------------------------------------
**      Class 선언
** ----------------------------------------------------------------
```

```
*/

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

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


/*
** ------------------------------------------------------------------
**      변수 선언
** ------------------------------------------------------------------
*/

        // 게임 상태.
        int statusTorus = TORUS_INTRO;
        int levelUp;                                    // 레벨 업
        int aniIntro;                                   // Intor Ani

        static int GAME_LEVEL = 9; // 초급 : 8, 중급 10, 고급 10;

        String nameNscore;                  // 점수
        XTextField xTextField;              // 이름 쓰기

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

/*
** ------------------------------------------------------------------
**      Image 선언
** ------------------------------------------------------------------
*/
```

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


        Image imgGameOver;                  // Game Over
        Image imgLevel;                     // 게임 레벨
        Image imgSNum;                      // 작은 숫자
        Image imgLNum;                      // 큰 숫자
        Image imgAllClear;          // All Clear


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



/*
** ----------------------------------------------------------------
**  Function:
**              - TorusCanvas
**
**  Description:
**              - 생성자.
**
**  Input:
**
**  Return value:
**
** ----------------------------------------------------------------
```

```
*/

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



                try
                {
                        // 이미지 최적화 방식이 있지만 교육상 편하게 하기 위해서
                        // 일부러 PNG로 읽었습니다.
                        // LBM으로 해서 Mixer를 이용해도 되고..
                        // 분할 로딩 해도 되고 여러가지 방법은 직접 테스트 해 보세요
                        // 실제적으로 이미지가 얼마 많지 않아서..그렇게 느리지는 않습니다.
                        imgIntro        = Image.createImage("/img/Intro.png");
        // 인트로
                        imgSlot         = Image.createImage("/img/Slot.png");
        // 바
                        imgTorus        = Image.createImage("/img/Torus.png");
        // 링
                        imgScrewBar     = Image.createImage("/img/ScrewBar.png");
        // 나사.
                        imgScrewPin     = Image.createImage("/img/ScrewPin.png");
        // 나사.
                        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



                        imgGameOver = Image.createImage("/img/Go.png");     // Game Over
                        imgLevel = Image.createImage("/img/Level.png");                 //
Level
```

```
                        imgSNum = Image.createImage("/img/SNum.png");                  //
Num
                        imgLNum = Image.createImage("/img/LNum.png");                  //
Num
                        imgAllClear = Image.createImage("/img/AllClear.png");   // All Clear

                }catch(Exception e){}



                // 이름 입력.
                xTextField = new XTextField("", MAX_STRING, TextField.ANY, this);
                xTextField.setBounds((XDisplay.width - 8 * Toolkit.MAX_CHARWIDTH) / 2,
LCD_Y + LCD_HEIGHT - 30,
                        8 * Toolkit.MAX_CHARWIDTH, Toolkit.FONT_HEIGHT + 1);

                xTimer = new XTimer(300);
                xTimer.setTimerListener(this);

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



/*
** ---------------------------------------------------------------------
**  Function:
**              - gameStart
**
**  Description:
**              - 게임 시작.(초기화)
**
**  Input:
**
**  Return value:
**
```

```
** ----------------------------------------------------------------
*/

        public void gameStart(int level)
        {
                GAME_LEVEL = level;
                levelUp = 0;
                tEngine.curLevel = 0;
                tEngine.scoreTorus = 0;

                // 첫번째 스테이지
                tEngine.statgeInit(0);
        }


/*
** ----------------------------------------------------------------
**  Function:
**                - gameOver
**
**  Description:
**                - 게임 Over(이름 쓰기)
**
**  Input:
**
**  Return value:
**
** ----------------------------------------------------------------
*/
        public void gameOver()
        {
                xTimer.cancel();
                statusTorus = TORUS_OVER;

                xTextField.setText("");
                xTextField.setFocus(true);
```

```
                    xTimer.setTime(300);

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


/*
** ---------------------------------------------------------------------
**   Function:
**             setRefreshArea
**
**   Description:
**             repaint 영역을 정한다.
**
**   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 한단계 짧아지는 조건
                        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);
```

```
                                        // Screw를 레벨업 시켜준다.
                                        tEngine.screwLevelUp();

                                        // Screw를 그려준다.
                                        tEngine.drawScrewTorus(Toolkit.graphics);
                                        break;

                                // 슬롯이 하나 더 늘어나는 조건.
                                case LEVEL_UP_CONDITION * 2:
                                        // 슬롯이 하나더 생길때만
                                        // 조건을 초기화 시킨다.
                                        levelUp = 0;

                                        // 슬롯을 하나더 추가시킨다.
                                        tEngine.slotLevelUp();

                                        // 시작할 준비.
                                        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)
                {
                        // 인트로 화면 보여주기.
                        case TORUS_INTRO:
                                g.drawImage(imgBG[aniIntro % 3], XDisplay.width / 2,
XDisplay.height2 / 2, g.HCENTER | g.VCENTER);

                                g2.drawImage((XDisplay.width - 110) / 2, LCD_Y + 10,
                                        imgIntro, 0, (aniIntro++ % 4) * 23, 110, 23,
Graphics2D.DRAW_COPY);

                                // 점수 보여주기.
                                if(nameNscore != null)
                                {
                                        g.setColor(0x000000);
                                        g.fillRect((XDisplay.width -
Toolkit.DEFAULT_FONT.stringWidth(nameNscore)) / 2 - 2,
                                                LCD_Y + 48,
Toolkit.DEFAULT_FONT.stringWidth(nameNscore) + 4, 19);
                                        g.setColor(0xffffff);
                                        g.drawString(nameNscore,
                                                (XDisplay.width -
Toolkit.DEFAULT_FONT.stringWidth(nameNscore)) / 2, LCD_Y + 50, g.LEFT | g.TOP);
                                }

                                g.drawImage(imgLevel, XDisplay.width / 2, LCD_Y + 90,
g.HCENTER | g.TOP);
                                break;
```

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

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

                                // Slot를 그린다.
                                tEngine.drawSlotTorus(g);

                                // Screw를 그린다.
                                tEngine.drawScrewTorus(g);

                                // 게임을 시작한다.
                                statusTorus = TORUS_GO;

                                // Repaint영역을 설저한다.
                                // Torus가 떨어지는 영역만 Repaint하면 된다.
                                setRefreshArea(tEngine.SLOT_X, 0,
                                        tEngine.SLOT_WIDTH, LCD_Y + LCD_HEIGHT);
                                break;

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

                        // 게임 오버.
                        case TORUS_OVER:
                                g.drawImage(imgGameOver,     XDisplay.width     /     2,
XDisplay.height2 / 2, g.HCENTER | g.VCENTER);
                                tEngine.drawScore(g, true);
```

```
                                xTextField.paint(g);
                                break;
                        }
                }


/*
** ----------------------------------------------------------------------
**  Function:
**          keyPressed
**
**  Description:
**          -  키 누름 처리
**
**  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)
                {
                        // 게임 레벨을 결정한다.
                        // 해당 레벨 및 기타등등의 값들은 원하시는 값대로 변경하시면 됩니다.
                        case TORUS_INTRO:
                                switch(keyCode)
                                {
                                        // 초보 모드
                                        case Canvas.KEY_NUM1:
                                                gameStart(6);

                                                xTimer.setTime(300);
                                                statusTorus = TORUS_READY;
                                                repaintArea();
```

```
                                        break;

                                // 중간 모드
                                case Canvas.KEY_NUM2:
                                        gameStart(9);

                                        xTimer.setTime(300);
                                        statusTorus = TORUS_READY;
                                        repaintArea();
                                        break;

                                // 고급 모드
                                case Canvas.KEY_NUM3:
                                        gameStart(12);

                                        xTimer.setTime(200);
                                        statusTorus = TORUS_READY;
                                        repaintArea();
                                        break;
                        }
                        break;

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

                // 게임 오버
                case TORUS_OVER:
                        switch(keyCode)
                        {
                                case Canvas.KEY_FIRE:
                                case Canvas.KEY_COMR:
                                        xTextField.setFocus(false);

                                        // 점수와 이름을 저장한다.
                                        if(nameNscore == null ||
```

```
                                                    tEngine.scoreTorus          >=
Integer.parseInt(nameNscore.substring(nameNscore.indexOf(":") + 1, nameNscore.length())))
                                                    {

            tMIDlet.writeInfo(xTextField.getText(), tEngine.scoreTorus);
                                                            nameNscore = tMIDlet.readInfo();
                                                    }

                                                    statusTorus = TORUS_INTRO;

                                                    xTimer.resume();
                                                    setRefreshArea(0,      0,      XDisplay.width,
XDisplay.height2);
                                                    repaintArea();
                                                    return;
                                            }

                                            if(xTextField.hasFocus())
                                                    xTextField.keyPressed(keyCode);
                                            break;
                        }
            }

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

```
*/

        public void keyReleased(int keyCode)
        {
        }
}
```

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 선언
** ------------------------------------------------------------------
*/

        // 실제적으로 이미지의 픽셀 크기로 정의 되어있다.
        // SLOT
        public final static int SLOT_UBAR_WIDTH      = 12;    // SLOT상단의 Torus가 들어가는 구멍 폭
        public final static int SLOT_UBAR_HEIGHT = 12;        // SLOT상단의 Torus가 들어가는 구멍 높이.

        public int SLOT_WIDTH;              // 전체 슬롯의 폭
        public int SLOT_HEIGHT;             // 전체 슬롯의 높이.
        public int SLOT_X;          // 슬롯의 X
        public int SLOT_Y;          // 슬롯의 Y
```

```
        // Screw
        // Bar
        public final static int SCREWBAR_WIDTH      = 11;    // 나사 못 바의 폭
        public final static int SCREWBAR_HEIGHT     = 4;             // 나사 못 바의 높이
        public int SCREWBAR_X ;                                      // 나사 못 바
의 X
        public int SCREWBAR_Y;                                       // 나사 못 바
의 Y

        // Pin
        public final static int SCREWPIN_WIDTH      = 11;    // 나사 못의 폭
        public int SCREWPIN_HEIGHT;                                  //  나사  못의
높이
        public int SCREWPIN_X ;                                      // 나사 못의 X
        public int SCREWPIN_Y;                                       // 나사 못의 Y

        // Torus
        public final static int TORUS_WIDTH         = 11;    // 하나의 Torus의 폭
        public final static int TORUS_HEIGHT                = 11;    // 하나의 Torus의 높이.
        public final static int TORUS_SLOT_HEIGHT   = 3;             // 슬롯에 들어간 Torus
의 높이.
        public final static int TORUS_NOMOTION              = 26;    // 슬롯에 들어갈 Torus
의 이미지의 Y좌표.
        public final static int TORUS_MOTION                        = 4;             // 4가
지 애니매이션을 가지고 있다.


        public final static int MAX_SCORE = 6;                       // 점수 자리수.

/*
** ----------------------------------------------------------------
**      Class 선언
** ----------------------------------------------------------------
*/
```

```
        TorusCanvas     tCanvas;

        // 떨어지는 Torus 객체
        // 원래는 해당 레벨 만큼만 가지고 있지만 메모리의 충분한 여유로 인하여
        // 전체 최대 개수를 생성한다. 그러므로 레벨이 올라가더라도 새로 생성없이 진행이 가능하다.
        // 솔직히 이것으로 얼마나 속도가 향상될지는 모르겠지만..무의미 할수도 있다. ㅜㅜ
        TorusRing[]     tRing = new TorusRing[16];



/*
** ------------------------------------------------------------------
**      변수 선언
** ------------------------------------------------------------------
*/

        // SLOT 관련.
        // 미리 배열을 최대 값으로 할당하여 속도를 빠르게 한다.
        // 단 이럴경우 해당 값들의 최대 최소 현재 위치등을 확실히 관리해야 한다.
        // 여기서 라인은 높이..즉 Torus가 쌓이는 개수를 의미한다.
        int[][] TORUS_SLOT = new int[12][16];       // SLOT의 2차원 배열.
        int[] slotCurLine = new int[16];                        // 현재 각각의 SLOT에 쌓여 있는 최대 라인.
        int[] slotDelLine = new int[16];                        // 지워져야할 라인을 기억한다.
        int slotMaxLine;                                                // SLOT의 가로 라인의 최대  개수.
        int slotCount;                                                  // SLOT의 개수

        // Screw 관련.
        int[] TORUS_SCREW = new int[12];  // 나사.(최대 개수)
        int screwCurStatck;                     // 스택의 현재 위치
        int maxOfscrewStack;                // 해당 레벨에 따른 Screw에 들어갈 Torus 최대 개수.
        int screwPos;                           // 나사 위치

        int curLevel;                                   // 현재 레벨..
```

```
        int scoreTorus;                 // 현재 점수.

        boolean bDelLine;               // 지워질 라인이 생겼다.
        boolean allClear;               // All Clear 했을 경우


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

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



/*
** ---------------------------------------------------------------------
**  Function:
**          initStage
**
**  Description:
**          스테이지를 초기화.
**
```

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

        public void statgeInit(int level)
        {
                // LEVEL 결정.
                curLevel = level;

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

                // SLOT에 들어갈 Torus 개수.
                slotMaxLine = TorusCanvas.TORUS_STAGE[curLevel][1];

                // Screw에 들어갈 Torus 개수.
                maxOfscrewStack = TorusCanvas.TORUS_STAGE[curLevel][2];

                // SLOT폭 초기화
                // 해당 SLOT폭을 이용하여 가운데 정렬위치의 X좌표를 구함.
                SLOT_WIDTH = SLOT_UBAR_WIDTH * slotCount + 1;
                SLOT_X = TorusCanvas.LCD_X + (TorusCanvas.LCD_WIDTH - SLOT_WIDTH) / 2;



                ///////// Screw 초기화.
                // Screw는 ㅗ 모양으로..
                // ㅗ = ㅣ(Pin) + ㅡ(Bar)  로 이미지가 따로 존재한다.
                // Screw Bar X, Y 좌표 설정
                screwCurStatck = 0;

                SCREWBAR_Y = TorusCanvas.LCD_Y + TorusCanvas.LCD_HEIGHT - 5 -
SCREWBAR_HEIGHT;
```

```
                SCREWBAR_X = SLOT_X + 1;

                // Screw Pin 초기화.
                // Screw 높이와 X, Y좌표 설정
                // 높이는 레벨에 따라서 다르다.
                SCREWPIN_HEIGHT = maxOfscrewStack * TORUS_SLOT_HEIGHT;
                SCREWPIN_Y = SCREWBAR_Y - SCREWPIN_HEIGHT;
                SCREWPIN_X = SCREWBAR_X;

                screwCurStatck = 0;

                // SLOT 높이 초기화
                SLOT_HEIGHT = SLOT_UBAR_HEIGHT + slotMaxLine * TORUS_SLOT_HEIGHT;//
SLOT에 들어가는 Torus의 높이는 3이다.
                SLOT_Y = SCREWPIN_Y - SLOT_HEIGHT;

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

                        tRing[i].setStaus(i);

                        // 현재 슬롯라인을 초기화 시킨다.
                        slotCurLine[i] = 0;
                }
        }


/*
** ----------------------------------------------------------------
**  Function:
**              - screwLevelUp
**
**  Description:
```

```
**                - 나사가 레벨업한다.(Torus를 쌓을수 있는 개수가 줄어든다.)
**
**  Input:
**
**  Return value:
**
** ---------------------------------------------------------------------
*/

        public void screwLevelUp()
        {
                // Screw에 들어갈 Torus 개수가 하나 줄어든다.
                maxOfscrewStack--;

                // Screw Bar 초기화.
                SCREWBAR_Y -= TORUS_SLOT_HEIGHT;

                // Screw Pin 초기화.
                SCREWPIN_HEIGHT -=  TORUS_SLOT_HEIGHT;

                // Screw에 Torus가 꽉 차있으면 맨위의 Torus를 버린다.
                if(screwCurStatck >= maxOfscrewStack)
                        screwCurStatck--;
        }

/*
** ---------------------------------------------------------------------
**  Function:
**                - slotLevelUp
**
**  Description:
**                - 슬롯이 레벨업한다.
**
**  Input:
**
**  Return value:
```

```
**
** ----------------------------------------------------------------
*/
	public void slotLevelUp()
	{
		statgeInit(++curLevel);
	}

/*
** ----------------------------------------------------------------
**  Function:
**		- slotPut
**
**  Description:
**		- Slot의 큐의 Put
**
**  Input:
**		- curSlot : 현재 선택된 슬롯.
**		- torusColor : 현재 슬롯에 들어갈 토로스 색상.
**
**  Return value:
**		- 성공, 실패.
**
** ----------------------------------------------------------------
*/

	public boolean slotPut(int curSlot, int torusColor)
	{
		// 슬롯이 꽉찬경우, 현재 슬롯 개수보다 많을경우, 최대 색상을 넘어갔을 경우
//		if(slotCurLine[curSlot] >= slotMaxLine || curSlot >= slotCurLine.length ||
//			torusColor > TorusCanvas.MAX_COLOR)
//			return false;

		// 실제적으로는 아래 에러만 검출하면 됨.
		if(slotCurLine[curSlot] >= slotMaxLine)
			return false;
```

```
                // SLOT에 쌓는다.
                TORUS_SLOT[curSlot][slotCurLine[curSlot]++] = torusColor;

                return true;
        }


/*
** ------------------------------------------------------------------
**  Function:
**              - slotGet
**
**  Description:
**              - Slot의 큐의 Get
**
**  Input:
**              - curSlot : 현재 선택된 슬롯.
**
**  Return value:
**              - 0 : 이상..발견..
**              - 1 Over : Color Value
**
** ------------------------------------------------------------------
*/

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

                // 실제적으로는 아래 에러만 검출하면 됨.
                if(TORUS_SLOT[curSlot][0] == 0)
                        return 0;
```

```
                // 맨 밑에 있는 Torus의 칼라값을 얻는다.
                int colorValue = TORUS_SLOT[curSlot][0];

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

                return colorValue;
        }



/*
** ------------------------------------------------------------------
**  Function:
**              - slotPush
**
**  Description:
**              - Slot의 스택의 Push
**
**  Input:
**              - curSlot : 현재 선택된 슬롯.
**              - torusColor : 현재 슬롯에 들어갈 토로스 색상.
**
**  Return value:
**              - 성공, 실패.
**
** ------------------------------------------------------------------
*/
        public boolean slotPush(int curSlot, int torusColor)
        {
                // 슬롯이 꽉찬경우, 현재 슬롯 개수보다 많을경우, 최대 색상을 넘어갔을 경우
//              if(slotCurLine[curSlot] >= slotMaxLine ||curSlot >= slotCurLine.length ||
//                      torusColor > TorusCanvas.MAX_COLOR)
//                      return false;
```

```
                // 실제적으로는 아래 에러만 검출하면 됨.
                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 스택의 Push
**
**  Input:
**              - curSlot : 현재 선택된 슬롯.
**
**  Return value:
**
** ---------------------------------------------------------------
*/

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

```
		// SLOT으로 부터 Get한다.
		int popColor = slotGet(curSlot);

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

/*
** ---------------------------------------------------------------------
**  Function:
**		- screwPop
**
**  Description:
**		- Screw 스택의 Pop
**
**  Input:
**		- curSlot : 현재 선택된 슬롯.
**
**  Return value:
**
** ---------------------------------------------------------------------
*/
	public void screwPop(int curSlot)
	{
		if(screwCurStatck <= 0)
			return;

		// SLOT으로 Push한다.
		if(slotPush(curSlot, TORUS_SCREW[screwCurStatck - 1]))
			TORUS_SCREW[--screwCurStatck] = 0;
	}


/*
** ---------------------------------------------------------------------
```

```
-------
**  Function:
**          deleteLine
**
**  Description:
**          - 한라인을 지운다.(현재 가장 적게 쌓인것을 중심으로 하면 가장 빨리지운다.)
**
**  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++)
                {
                        // 만일 한슬롯이라도 비면 빠져나간다.
                        if(slotCurLine[0] < 1)
                                return bDel;
                        else
                        {
                                // 그렇지 않으면 가장 적게 쌓여있는 슬롯을 알아낸다.
                                if(slotCurLine[i] < countMinSlot)
                                        countMinSlot = slotCurLine[i];
                        }
                }
```

```
                // 가정 적게 있는 슬롯만큼만 비교하면 된다.
                for(i = 0, k = 0; i < countMinSlot; i++)
                {
                        // 첫번째 슬롯에 있는 색상을 비교한다.
                        torusColor = TORUS_SLOT[0][i];
                        bDel = true;
                        slotDelLine[k] = -1;

                        for(j = 0; j < slotCount; j++)
                        {
                                // 색상이 다른게 있으면 다음 색상으로 넘어간다.
                                if(torusColor != TORUS_SLOT[j][i])
                                {
                                        bDel = false;
                                        break;
                                }
                        }

                        // 같은 것이 있을경우에는 지워질 라인을 등록한다.
                        if(bDel)
                                slotDelLine[k++] = i;
                }

                // 지워질 라인이 존재 한다면..
                if(k > 0)
                {
                        // 지운다.
                        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]);

                                // Torus가 지원지고 난후에 레벨에 따른 점수를 매긴다.
                                scoreTorus += ((curLevel + 1) * 10 *
TorusCanvas.GAME_LEVEL);
```

```
                        }

                        allClear = true;

                        // All Clear는 전체 Torus가 사라졌우를 의미하면
                        // 점수가 더 높다.
                        // 이럴경우 화면에 All Clear를 보여주면 좋다.
                        // 현재는 이미지만 있다. 구현은 직접 해보심 좋을듯 하다.
                        for(i = 0; i < slotCount; i++)
                        {
                                if(screwCurStatck > 0 || slotCurLine[i] > 0)
                                {
                                        allClear = false;
                                        break;
                                }
                        }

                        if(allClear)
                                scoreTorus += (curLevel + 1) * 100;

                        bDel = true;
                }else
                        bDel = false;

                return bDel;
        }




/*
** ---------------------------------------------------------------
**  Function:
**          drawSlotTorus
**
**  Description:
```

```
**            Slot, Torus를 그린다.
**
**  Input:
**
**  Return value:
**
** ----------------------------------------------------------------
*/

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

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

                // drawSlot
                // 처음에는 한가지 이미지를 가지고 for문을 돌려서 그릴려고 했지만
                // 메모리를 많이 요하지 않는 게임이라서 편하게 했습니다.
                tCanvas.g2.drawImage(SLOT_X, SLOT_Y,
                        tCanvas.imgSlot, 0, 0,
                        SLOT_WIDTH, SLOT_HEIGHT, Graphics2D.DRAW_COPY);

                // drawTorus
                // 해당 Slot정보(2차원 배열)를 가지고 안에 쌓여 있는 Torus를 그린다.
                for(i = 0; i < slotCount; i++)
                {
                        // 해당 슬롯에 쌓여있는 Torus개수.
                        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, Torus를 그린다.
**
**  Input:
**
**  Return value:
**
** ---------------------------------------------------------------------
*/

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

                    // Back Color
                    // Screw가 움직이는 하단을 검은색 바탕으로 칠한다.
                    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 에 쌓여있는 Torus를 그린다.
                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:
**          drawScore
**
**  Description:
**          점수를 그린다.
**
**  Input:
**              - bSize : true : 큰 숫자 점수, false : 작은 숫자 점수.
**
```

```
**   Return value:
**
** ----------------------------------------------------------------
*/

        public void drawScore(Graphics g, boolean bSize)
        {
                int score = scoreTorus;

                if(bSize)
                {
                        for(int i = 0; i < MAX_SCORE; i++, score /= 10)
                                tCanvas.g2.drawImage(TorusCanvas.LCD_X                  +
TorusCanvas.LCD_WIDTH   / 2 + 17 - 9 * i, TorusCanvas.LCD_Y + 30,
                                        tCanvas.imgLNum,  9  *  (score  %  10),  0,  9,  12,
Graphics2D.DRAW_COPY);
                }else
                {
                        for(int i = 0; i < MAX_SCORE; i++, score /= 10)
                                tCanvas.g2.drawImage(TorusCanvas.LCD_X                  +
TorusCanvas.LCD_WIDTH   / 2 + 11 - 6 * i, TorusCanvas.LCD_Y + 2,
                                        tCanvas.imgSNum,  6  *  (score  %  10),  0,  6,  8,
Graphics2D.DRAW_COPY);
                }
        }



/*
** ----------------------------------------------------------------
**   Function:
**           paint
**
**   Description:
**
**
**   Input:
```

```
**
**   Return value:
**
** ----------------------------------------------------------------
*/

        public void paint(Graphics g)
        {
                // 삭제되는 라인이 있을 경우
                if(bDelLine)
                {
                        // 삭제가 되면 Slot를 다시 그린다.
                        if(delLine())
                                drawSlotTorus(g);

                        bDelLine = false;
                }

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

                                // Torus가 움직일 때
                                // 쌓이거나..할때만 Torus가 지워지는 조건이 된다.
                                case 1:
                                        bDelLine = true;

                                        // Torus가 움직일 때 그리는 조건 성립.
                                        // Slot 그리기.
                                        drawSlotTorus(g);
```

```
                                        break;
                        }
                }

                // Score 그리기.
                drawScore(g, false);
        }

/*
** ---------------------------------------------------------------
**  Function:
**          keyPressed
**
**  Description:
**
**
**  Input:
**
**  Return value:
**
** ---------------------------------------------------------------
*/
        public void keyPressed(int keyCode)
        {
                // 상하좌우 키가 느릴경우에는 XDisplay.refresh()를 사용한다.
                switch(keyCode)
                {
                        // Screw에 있는 것을 SLOT으로 올린다.
                        case Canvas.KEY_UP://     = 141;
                                screwPop(screwPos);

                                // Torus를 위아래로 움직이거나 할 때
                                // 지워지는 조건이 성립한다.
                                bDelLine = true;

                                // 속도면에서 좌우상하를 움직일 경우 체감속도가 빠름
```

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

                        // Screw를 좌로 움직인다.
                        // 맨 왼쪽으로 갈 경우에는 더 이상 가지 못한다.
                        case Canvas.KEY_LEFT://   = 142;
                                screwPos = (--screwPos < 0)?0:screwPos;

                                // 속도면에서 좌우상하를 움직일 경우 체감속도가 빠름
                                drawScrewTorus(Toolkit.graphics);
                                XDisplay.refresh(SCREWPIN_X,  SCREWPIN_Y,  SLOT_WIDTH,
SCREWPIN_HEIGHT + SCREWBAR_HEIGHT);
                                break;

                        // Screw를 우로 움직인다.
                        // 맨 오른쪽으로 갈 경우에는 더 이상 가지 못한다.
                        case Canvas.KEY_RIGHT:// = 145;
                                screwPos = (++screwPos >= slotCount)?
                                        slotCount - 1:screwPos;

                                // 속도면에서 좌우상하를 움직일 경우 체감속도가 빠름
                                drawScrewTorus(Toolkit.graphics);
                                XDisplay.refresh(SCREWPIN_X,  SCREWPIN_Y,  SLOT_WIDTH,
SCREWPIN_HEIGHT + SCREWBAR_HEIGHT);
                                break;

                        // SLOT에 있는 것을 Screw로 내린다.
                        case Canvas.KEY_DOWN:// = 146;
                                screwPush(screwPos);

                                // Torus를 위아래로 움직이거나 할 때
                                // 지워지는 조건이 성립한다.
                                bDelLine = true;
```

```
                                    // 속도면에서 좌우상하를 움직일 경우 체감속도가 빠름
                                    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
{
	int x;				// Torus X축
	int y;				// Torus Y축
	int color;		// Torus 색상.
	int motion;			// Torus 애니
	int selSlot;			// 해당 Torus의 슬롯.
	int speed;			// Torus가 떨어지는 스피드

	Random random;
	TorusCanvas tCanvas;



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

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

```
/*
** ------------------------------------------------------------------
**  Function:
**          setStaus
**
**  Description:
**          상태를 초기화 시킨다.
**
**  Input:
**
**  Return value:
**
** ------------------------------------------------------------------
*/
        public void setStaus(int selSlot)
        {
                // 슬롯을 선택
                this.selSlot = selSlot;

                // 애니매이션을 처음으로
                motion = 0;

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

                // 색상, 레벨 마다 떨어지는 스피드가 다르다.
                speed = ((color + TorusCanvas.GAME_LEVEL) * TorusCanvas.MAX_COLOR) %
TorusCanvas.GAME_LEVEL + TorusCanvas.GAME_LEVEL / 2;

                // X, Y 축
                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)
        {
                // 이전 Torus에 위치에 영역 만큼 바탕색으로 지운다.
                g.setColor(0x000000);
                g.fillRect(x, y, TorusEngine.TORUS_WIDTH, TorusEngine.TORUS_HEIGHT);

                // 떨어지는 간격을 조절한다. 범위가 크면 그만큼 빨리 떨어진다.
                y += speed;

                // 해당 색상과 현재 애니매이션에 맞는 Torus를 그린다.
                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 슬롯에 도착하면..
                    if(y >= tCanvas.tEngine.SLOT_Y)
                    {
                              // 큐가 넘치면 게임 오버..
                              if(!tCanvas.tEngine.slotPut(selSlot, color + 1))
                                        return -1;

                              // 다시 위로 올린다.
                              setStaus(selSlot);
                              return 1;
                    }

                    return 0;
          }
}
```

다음주 강좌는 AI 즉 인공지능 입니다. 해당 Torus와 대전을 해보고 싶었던 저는 AI를 만들었습니다. AI란 무엇인가? 라고 한다면 모든 경우의 수를 프로그래밍 한다고 생각하시면 됩니다. 그래서 다음 주에는 Torus의 AI에 대해서 말씀드리겠습니다. AI만들려고 프로그램 하나 더 만들어서 노가다점 했습니다. AI가 슬롯을 3개만 있을 경우만 만들었습니다. 3개만 가지고 만드는데도 장난이 아니더군요. 실제로 혼자서 플레이하게 내버려 두었는데 죽지를 않습니다. 잘만들었다고 생각이 듭니다..ㅎㅎㅎ 가끔씩 운나쁘면 죽는데..다 고쳤습니다. 저보다 더 잘할 듯. ㅜㅜ

여튼 좋은 하루 되시고요..이 강좌가 끝나면..좋은 Torus게임이 나오면 좋겠네요..^^
ㄴ
닉네임 : 핑크레드
성격 : 왕소심, 왕삐짐, 이중인격, 수다쟁이
이메일 : pinkred@hanafos.com

추신 : 강좌를 다른 사이트에 올리시는 것은 자유입니다..단지 저한테 메일 한 통 이라도 보내주시고 올리시면 감사하겠습니다. 언제나 좋은 하루 되십시오.