



ANDROID STUDIO -SURFACE & THREAD-

SOULSEEK

1. SURFACE

- **View Class**의 일종이다. → **WinAPI**의 더블 버퍼링과 같은 개념
- **onDraw()**와 **invalidate()**가 필요 없다
- 독립적인 그리기 작업을 할 수 있는 공간
- **Thread** 설치가 필요하다
 - **Main Thread**에서 모든 일을 처리하는 것보다 그리기 전용 고속 메모리 버퍼인 **Surface**를 이용하는 것이 게임 제작에 더 적합하다.

Surface는 **Surface(메모리 버퍼)**와 **SurfaceHolder**로 이루어져 있다.

SurfaceView

Surface

고속메모리(버퍼)이며 **lockCanvas** 메소드 에 의해서 사용자의 문자나 이미지가 그려지게 되는 곳이다.

SurfaceHolder

SurfaceHolder는 서피스뷰에서 전용으로 사용되는 인터페이스이다. **lockCanvas()**와 **unlockCanvasAndPost()**를 가지고 있다.

<https://developer.android.com/reference/android/view/SurfaceHolder.html> 에서 해당 **API**에 관련된 부분들을 조금 더 파악할 수 있다.

1. SURFACE

SurfaceHolder

- **SurfaceView**에서 전용으로 사용되는 인터페이스.
 - **Thread**를 이용해서 **Surface**에 한번에 그려 놓고 **MainView**에 전송한다.
- **Surface**에 행하는 동작(그림 그리기, 문자등) 및 **Surface**에 진행된 작업을 실제 사용자 기기의 **MainView**로 복사하는 역할을 한다.
- **mHolder = getHolder();** 의 형식으로 얻어올 수 있다.
- 메모리에서 진행되는 작업이기 때문에 사용하는 구간에 대한 **Lock**이 걸려서 보호를 해야하기 때문에 그림, 문자 출력 메소드를 진행하게 되면 **lockCanvas()** 메소드를 사용해야한다.
- 모든 작업을 완료하게 되면 **unlockCanvasAndPost()** 메소드를 호출하여 **lock**를 해제하고 실제 **MainView**에 **SurfaceView**를 복사한다
- **lockCanvas()**, **unlockCanvasAndPost()** 각 메소드는 **Thread** 진행 메소드인 **run()**함수 안에 존재해야 한다.

unlockCanvasAndPost() 메소드는 서피스에 진행하는 행동을 멈추고 실제화면으로 복사한다.

lockCanvas() 메소드를 이용해서 서피스에 **lock**을 걸고 동작을 수행한다.

View(실제화면)

SurfaceView

THREAD

- 응용프로그램이 구동되고 있는 작업공간.
- **Main Thread**에서 모든 처리를 기다리면서 넘어가기 힘들다면 나누어라.
- **Surface**와 연결하여 그리는 부분과 인터페이스 동작부분을 나누어 줄 수 있다.

Thread에 SurfaceView 연결하기

1. 서피스뷰를 상속받고 **SurfaceHolder.Callback**을 구현하는 클래스를 만든다.
 - **public class MySurface extends Surface implements SurfaceHolder.Callback**
2. 만든 **Surface** 클래스안에 **Thread**를 상속받은 내부 클래스를 만든다.
 - **Class MyThread extends Thread**
3. 만든 **Surface** 클래스 멤버 필드로 **Thread** 객체 및 **Surface Holder**를 선언한다.
 - **MyThread myThread;** // 쓰레드 객체 선언
 - **SurfaceHolder mHolder;** //서피스 홀더 객체 선언
4. 만든 **Surface** 클래스의 생성자 안에서 **Thread**를 생성한다. 이때 생성자 인수로 **SurfaceHolder**객체 및 **Context**객체를 넘겨준다.
 - **myThread = new Thread(holder, context);**
5. **surfaceCreated()** 메소드 안에 **start()**메소드를 넣는다.(쓰레드를 가동한다)
 - **myThread.start();**

THREAD

6. 만든 **Tread** 클래스 안에 **run()**메소드를 만들고 안에서 다음과 같이 촬영하면 된다.

```
public void run(){
    Canvas canvas = null;    //캔버스객체를 생성한다. 캔버스는 도화지이다.
    while(true){             //반복실행을 한다.
        canvas = mHolder.lockCanvas();    //서피스에 픽셀을 나타낼 수 있다.

        try{
            synchronized(mHolder){
                drawEverything(canvas);    //모든 픽셀을 서피스에 표현한다.
            }
        }
        finally{
            if(canvas != null){
                mHolder.unlockCanvasAndPos(canvas); //실제화면에 복사
            }
        }
    }
}
```

THREAD

파일들의 연결.

The screenshot displays four files in an Android Studio IDE, illustrating the connections between them:

- AndroidManifest.xml**: Contains the `<activity android:name=".MainActivity">` tag. A red box highlights this tag, and a blue arrow points from it to the `MainActivity.java` file.
- MainActivity.java**: Contains the `public class MainActivity extends AppCompatActivity` declaration. A red box highlights `MainActivity`, and a blue arrow points from it to the `MySurface.java` file.
- MySurface.java**: Contains the `public class MySurface extends SurfaceView implements SurfaceHolder.Callback` declaration. A red box highlights `MySurface`, and a blue arrow points from it to the `activity_main.xml` file.
- activity_main.xml**: Contains the `<com.soulseek.surfacegame.MySurface` tag. A red box highlights this tag, and a blue arrow points from it to the `MySurface.java` file.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.soulseek.surfacegame">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="surfaceGame"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN">
                <category android:name="android.intent.category.LAUNCHER">
            </intent-filter>
        </activity>
    </application>
</manifest>
```

```
package com.soulseek.surfacegame;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

```
package com.soulseek.surfacegame;

import ...

public class MySurface extends SurfaceView implements SurfaceHolder.Callback
{
    //SurfaceView
    MyThread mThread;
    SurfaceHolder mHolder;
    Context mContext;

    Bitmap screen;
    int Width, Height;
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <com.soulseek.surfacegame.MySurface
        android:id="@+id/mySurface"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
    />
</LinearLayout>
```


THREAD

SurfaceView 구현

```
MySurface.java
25
26 public MySurface(Context context, AttributeSet attrs)
27 {
28     super(context, attrs);
29     SurfaceHolder holder = getHolder();
30     holder.addCallback(this);
31
32     mThread = new MyThread(holder, context);
33     InitApp();
34 }
35
36 //App 초기화
37 private void InitApp()
38 {
39     Display display
40         = ((WindowManager) mContext.getSystemService(Context.WINDOW_SERVICE)).getDefaultDisplay();
41
42     Width = display.getWidth();
43     Height = display.getHeight();
44
45     screen = BitmapFactory.decodeResource(getResources(), R.drawable.screen);
46     screen = Bitmap.createScaledBitmap(screen, Width, Height, filter: true);
47 }
```

```
ySurface.java
@Override
public void surfaceCreated(SurfaceHolder holder)
{
    //Thread Start
    mThread.start();
}

@Override
public void surfaceChanged(SurfaceHolder arg0, int format, int width, int height)
{}

@Override
public void surfaceDestroyed(SurfaceHolder holder)
{}
```

```
MySurface.java
68 //Thread Class
69 public class MyThread extends Thread
70 {
71     public MyThread(SurfaceHolder holder, Context context)
72     {
73         mHolder = holder;
74         mContext = context;
75
76         //연산 부분
77     }
78
79 @
80 public void drawEverything(Canvas canvas)
81 {
82     Paint p1 = new Paint();
83     p1.setColor(Color.RED);
84     p1.setTextSize(50);
85     canvas.drawBitmap(screen, left: 0, top: 0, p1);
86 }
```

Surface View를 구현하는 부분.
기본적으로 Override해줘야하는 경우가 많
기때문에 Override 함수를 모두 기입해줘야
하며, Thread를 만들어서 작업을 진행 하게
만들어야 한다.

THREAD

run() 구현

```
MySurface.java x
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110

public void run()
{
    Canvas canvas = null;
    while(true)
    {
        canvas = mHolder.lockCanvas();

        try
        {
            synchronized(mHolder)
            {
                drawEverything(canvas);
            }
        }
        finally
        {
            if(canvas != null)
            {
                mHolder.unlockCanvasAndPost(canvas);
            }
        }
    }
}
```

run() 메소드에서 Canvas를 lockCanvas 인스턴스로 생성해서 drawEverything() 메소드에 넘겨서 Canvas에 그리는 작업을 수행하게 한다. 그 후 모든 작업이 진행되어서 코드블록으로 진행되면 finally에 들어가게 되고 canvas가 null이 아닐 경우에(그려졌을 경우에) unlockCanvasAndPost() 메소드에 넘겨줘서 실제 View에 그리게 한다.