001 프로젝트 생성

AndroidProject4BoardApp

002 ViewBinding 설정

```
[build.gradle.kts]

buildFeatures{
viewBinding = true
}
```

```
[ MainActivigty.kt ]

lateinit var activityMainBinding: ActivityMainBinding

override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)

activityMainBinding = ActivityMainBinding.inflate(layoutInflater)
setContentView(activityMainBinding.root)
}
```

003 SplashScreen 구성하기

google 가이드 **https://developer.android.com/develop/ui/views/launch/splash-screen?hl=ko**

라이브러리 설정

```
[ build.gradle.kts ]

implementation("androidx.core:core-splashscreen:1.0.1")
```

사용할 아이콘 이미지를 준비한다.

```
1. File > New > Image Asset 메뉴를 선택한다.
2. Name : ic_logo
3. Foreground Layer : 전면에 보이는 이미지를 설정한다.
  - Layer name : foreground_logo
  - Asset Type : Text
  - Text : Lion
4. Background Layer : 배경
  - layer name : background_logo
  - Asset type : Color
  - Color : 3DDC84
```

themes.xml 에 Splash Screen 테마를 구성해준다

- windowSplashScreenBackground : SplashScreen 의 배경을 설정한다. 색상값이나 이미지 등을 사용할 수 있다.
- windowSplashScreenAnimationIcon : SplashScreen에서 사용할 아이콘을 설정한다.
- windowSplashScreenAnimationDuration : SplashScreen을 보여줄 시간을 설정한다. 최대 1000ms
- postSplashScreenTheme : SplashScreen을 보여주고 나서 다음에 나올 화면의 테마를 설정해준다. AndoridManifest.xml 에 application 태그의 android:theme 속성에 있는 것을 설정해주면 된다.

```
[ res/values/tehems.xml ]

<!-- Splash Screen Theme -->
<style name="AppTheme.Starting" parent="Theme.SplashScreen">
<item name="windowSplashScreenBackground">@color/background_logo</item>
<item name="windowSplashScreenAnimatedIcon">@mipmap/ic_logo</item>
<item name="windowSplashScreenAnimationDuration">1000</item>
<item name="postSplashScreenTheme">@style/Theme.AndroidProject4BoardApp</item>
</style>
```

AndroidManifest.xml 의 기본 테마를 Splash Screen 테마로 변경해준다.

```
[ AndroidManifest.xml ]
<application
(생략략)
android:theme="@style/AppTheme.Starting"
tools:targetApi="31">
```

스플래시 스크린이 나타나도록 코드를 작성해준다.

```
[ MainActivity - onCreate]

// 스플래쉬 스크린이 나타나게 한다.
// 반드시 setContentView 전에 해야 한다.
installSplashScreen()
```

004 MainActivity에 프래그먼트 사용 준비

```
[ activity_main.xml ]

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity" >

    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/containerMain"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

여러가지들을 정의할 Tools.kt 생성

```
[ Tools.kt ]

class Tools {

    companion object{

    }
}
```

프래그먼트의 이름들을 담을 enum class 작성

```
[ Tools.kt ]

// MainActivity에서 보여줄 프래그먼트들의 이름
enum class MainFragmentName(var str:String){
    A_FRAGMENT("A"),
    B_FRAGMENT("B"),
}
```

MainActivity 에 프래그먼트 주소값을 담을 프로퍼티 정의

```
[ MainActivity.kt ]

    // 프래그먼트의 주소값을 담을 프로퍼티
    var oldFragment:Fragment? = null
    var newFragment:Fragment? = null
```

프로그먼트 기본 코드를 붙혀 넣는다.

replaceFragment 의 첫 번째 매개변수 타입을 MainFragmentName 로 변경한다.

FragmentContainerView의 아이디를 R.id.containerMain 로 변경한다.

removeFragment 메서드의 첫 번째 매개변수의 타입을 MainFragmentName 으로 변경한다.

```
[ MainActivity.kt ]

    // 지정한 Fragment를 보여주는 메서드
    // name : 프래그먼트 이름
    // addToBackStack : BackStack에 포함 시킬 것인지
    // isAnimate : 애니메이션을 보여줄 것인지
    // data : 새로운 프래그먼트에 전달할 값이 담겨져 있는 Bundle 객체
    fun replaceFragment(name:MainFragmentName, addToBackStack:Boolean, isAnimate:Boolean, data:Bundle?){

        SystemClock.sleep(200)

        // Fragment를 교체할 수 있는 객체를 추출한다.
        val fragmentTransaction = supportFragmentManager.beginTransaction()
```

```
// oldFragment에 newFragment가 가지고 있는 Fragment 객체를 담아준다.
if(newFragment != null){
    oldFragment = newFragment
}

// 이름으로 분기한다.
// Fragment의 객체를 생성하여 변수에 담아준다.
when(name){

}

// 새로운 Fragment에 전달할 객체가 있다면 arguments 프로퍼티에 넣어준다.
if(data != null){
    newFragment?.arguments = data
}

if(newFragment != null){

    // 애니메이션 설정
    if(isAnimate == true){
        // oldFragment -> newFragment
        // oldFragment : exitTransition
        // newFragment : enterTransition

        // newFragment -> oldFragment
        // oldFragment : reenterTransition
        // newFragment : returnTransition

        // MaterialSharedAxis : 좌우, 위아래, 공중 바닥 사이로 이동하는 애니메이션 효과
        // X - 좌우
        // Y - 위아래
        // Z - 공중 바닥
        // 두 번째 매개변수 : 새로운 화면이 나타나는 것인지 여부를 설정해준다.
        // true : 새로운 화면이 나타나는 애니메이션이 동작한다.
        // false : 이전으로 되돌아가는 애니메이션이 동작한다.

        if(oldFragment != null){
            // old에서 new가 새롭게 보여질 때 old의 애니메이션
            oldFragment?.exitTransition = MaterialSharedAxis(MaterialSharedAxis.X, true)
            // new에서 old로 되돌아갈때 old의 애니메이션
            oldFragment?.reenterTransition = MaterialSharedAxis(MaterialSharedAxis.X, false)
```

```
                oldFragment?.enterTransition = null
                oldFragment?.returnTransition = null
            }

            // old에서 new가 새롭게 보여질 때 new의 애니메이션
            newFragment?.enterTransition = MaterialSharedAxis(MaterialSharedAxis.X, true)
            // new에서 old로 되돌아갈때 new의 애니메이션
            newFragment?.returnTransition = MaterialSharedAxis(MaterialSharedAxis.X, false)

            newFragment?.exitTransition = null
            newFragment?.reenterTransition = null
        }

        // Fragment를 교체한다.(이전 Fragment가 없으면 새롭게 추가하는 역할을 수행한다)
        // 첫 번째 매개 변수 : Fragment를 배치할 FragmentContainerView의 ID
        // 두 번째 매개 변수 : 보여주고하는 Fragment 객체를
        fragmentTransaction.replace(R.id.containerMain, newFragment!!)

        // addToBackStack 변수의 값이 true면 새롭게 보여질 Fragment를 BackStack에 포함시켜 준다.
        if(addToBackStack == true){
            // BackStack 포함 시킬때 이름을 지정해주면 원하는 Fragment를 BackStack에서 제거할 수 있다.
            fragmentTransaction.addToBackStack(name.str)
        }
        // Fragment 교체를 확정한다.
        fragmentTransaction.commit()
    }
}

// BackStack에서 Fragment를 제거한다.
fun removeFragment(name:MainFragmentName){
    SystemClock.sleep(200)

    // 지정한 이름으로 있는 Fragment를 BackStack에서 제거한다.
    supportFragmentManager.popBackStack(name.str, FragmentManager.POP_BACK_STACK_INCLUSIVE)
}
```

005 LoginFragment 작업

fragment 라는 이름의 패키지를 생성해준다.

fragment 패키지에 LoginFragment 를 생성해준다.

Fragment의 이름을 정의해준다.

```
[Tools.kt]

// MainActivity에서 보여줄 프래그먼트들의 이름
enum class MainFragmentName(var str:String){
    LOGIN_FRAGMENT("LoginFragment"),
    B_FRAGMENT("B"),
}
```

프래그먼트 생성 코드를 작성해준다.

```
[ MainActivity - replaceFragment]

        // 이름으로 분기한다.
        // Fragment의 객체를 생성하여 변수에 담아준다.
        when(name){
            // 로그인 화면
            MainFragmentName.LOGIN_FRAGMENT -> {
                newFragment = LoginFragment()
            }
            MainFragmentName.B_FRAGMENT -> {

            }
        }
```

LoginFragment가 나타나도록 한다.

```
[ MainActivity - onCreate ]

        // 첫 화면을 띄워준다.
        replaceFragment(MainFragmentName.LOGIN_FRAGMENT, false, false, null)
```

제공한 이미지 파일들을 drawable 폴더에 넣어준다.

fragment_login.xml 에 화면 요소를 배치한다.

```
[ fragment_login.xml ]
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:transitionGroup="true"
    tools:context=".fragment.LoginFragment" >

    <com.google.android.material.appbar.MaterialToolbar
        android:id="@+id/toolbarLogin"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="?attr/colorPrimary"
        android:minHeight="?attr/actionBarSize"
        android:theme="?attr/actionBarTheme" />

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical"
            android:padding="10dp" >

            <com.google.android.material.textfield.TextInputLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:hint="아이디"
                app:endIconMode="clear_text"
                app:startIconDrawable="@drawable/person_24px">

                <com.google.android.material.textfield.TextInputEditText
                    android:id="@+id/textFieldLoginUserId"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:inputType="text"
                    android:textAppearance="@style/TextAppearance.AppCompat.Large" />
            </com.google.android.material.textfield.TextInputLayout>
```

```
        <com.google.android.material.textfield.TextInputLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            android:hint="비밀번호"
            app:endIconMode="password_toggle"
            app:startIconDrawable="@drawable/key_24px">

            <com.google.android.material.textfield.TextInputEditText
                android:id="@+id/textFieldLoginUserPw"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:inputType="text|textPassword"
                android:textAppearance="@style/TextAppearance.AppCompat.Large" />
        </com.google.android.material.textfield.TextInputLayout>

        <com.google.android.material.checkbox.MaterialCheckBox
            android:id="@+id/checkBoxLoginAuto"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            android:text="자동로그인"
            android:textAppearance="@style/TextAppearance.AppCompat.Large" />

        <Button
            android:id="@+id/buttonLoginSubmit"
            style="@style/Widget.Material3.Button.OutlinedButton"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            android:text="로그인"
            android:textAppearance="@style/TextAppearance.AppCompat.Large" />

        <com.google.android.material.divider.MaterialDivider
            android:id="@+id/view"
            android:layout_width="match_parent"
            android:layout_height="2dp"
            android:layout_marginTop="10dp"
            android:background="@color/black" />

        <Button
            android:id="@+id/buttonLoginJoin"
            style="@style/Widget.Material3.Button.OutlinedButton"
```

```
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginTop="10dp"
                android:text="회원가입"
                android:textAppearance="@style/TextAppearance.AppCompat.Large" />

        </LinearLayout>
    </ScrollView>
</LinearLayout>
```

theme.xml 에 색상을 커스터마이징 해준다

```
[ res/values/teheme.xml ]

        <!-- 모든 UI 요소의 기본 색상 -->
        <item name="colorPrimary">@color/material_dynamic_primary40</item>
        <!-- 상단 상태바 색상 -->
        <item name="colorPrimaryDark">@color/material_dynamic_tertiary99</item>
        <!-- 상단 상태바에 나타나는 정보를 보여주도록 설정한다. 상단 상태바의 색상이 밝은 색인지 여부를 설정한다 -->
        <item name="android:windowLightStatusBar">true</item>


    <!-- Toolbar 용 테마. Toolbar 마다 지정해준다. -->
    <style name="Theme.AndroidProject4BoardApp.Toolbar" parent="ThemeOverlay.Material3.Toolbar.Surface">
        <item name="colorPrimary">@color/material_dynamic_tertiary99</item>
    </style>
```

fragment_login.xml 의 툴바에 스타일을 지정해준다.

```
[ res/layout/fragment_login.xml ]

    <com.google.android.material.appbar.MaterialToolbar
        android:id="@+id/toolbarLogin"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:minHeight="?attr/actionBarSize"
        android:theme="?attr/actionBarTheme" />
```

LoginFragmeht 의 기본 코드를 작성해준다.

```
[ LoginFragment.kt ]

    lateinit var fragmentLoginBinding: FragmentLoginBinding
    lateinit var mainActivity: MainActivity

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {
        // Inflate the layout for this fragment

        fragmentLoginBinding = FragmentLoginBinding.inflate(inflater)
        mainActivity = activity as MainActivity

        return fragmentLoginBinding.root
    }
```

툴바 설정하는 메서드를 작성해준다.

```
[LoginFragmeht.kt]

    // 툴바 설정
    fun settingToolbar(){
        fragmentLoginBinding.apply {
            toolbarLogin.apply {
                // 타이틀
                title = "로그인"
            }
        }
    }
```

onCreateView에서 호출해준다.

```
[LoginFragment.kt - onCreateView]

        settingToolbar()
```

006 JoinFragment 작업

JoinFragment 를 생성한다.

Fragment 의 이름을 등록해준다.

```
[ Tools.kt ]

// MainActivity에서 보여줄 프래그먼트들의 이름
enum class MainFragmentName(var str:String){
    LOGIN_FRAGMENT("LoginFragment"),
    JOIN_FRAGMENT("JoinFragment"),
}
```

Fragment 객체 생성 코드를 넣어준다.

[MainActivity.kt - repalceFragment]

```
        when(name){
            (생략)
            // 회원가입 화면 1
            MainFragmentName.JOIN_FRAGMENT -> {
                newFragment = JoinFragment()
            }
        }
```

LoginFragment 에 버튼의 리스너를 구현해주고 JoinFragment로 이동할 수 있도록 한다.

```
[LoginFragment.kt]

    // 회원 가입 버튼
    fun settingButtonLoginJoin(){
        fragmentLoginBinding.apply {
            buttonLoginJoin.apply {
                // 버튼을 눌렀을 때
                setOnClickListener {
                    // JoinFragment가 보여지게 한다.
                    mainActivity.replaceFragment(MainFragmentName.JOIN_FRAGMENT, true, true, null)
                }
            }
        }
    }
```

메서드를 호출한다.

```
[LoginFragment.kt - onCreateView]

        settingButtonLoginJoin()
```

fragment_join.xml 에 화면을 구성해준다.

```
[res/layout/fragment_join.xml]

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:transitionGroup="true"
    tools:context=".fragment.JoinFragment" >

    <com.google.android.material.appbar.MaterialToolbar
        android:id="@+id/toolbarJoin"
        style="@style/Theme.AndroidProject4BoardApp.Toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:minHeight="?attr/actionBarSize"
        android:theme="?attr/actionBarTheme" />
```

```
    android:theme="?attr/actionBarTheme" />

<ScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:padding="10dp" >

        <com.google.android.material.textfield.TextInputLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="아이디"
            app:endIconMode="clear_text"
            app:startIconDrawable="@drawable/person_24px">

            <com.google.android.material.textfield.TextInputEditText
                android:id="@+id/textFieldJoinUserId"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:inputType="text"
                android:textAppearance="@style/TextAppearance.AppCompat.Large" />
        </com.google.android.material.textfield.TextInputLayout>

        <Button
            android:id="@+id/buttonJoinCheckId"
            style="@style/Widget.Material3.Button.OutlinedButton"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            android:text="중복확인"
            android:textAppearance="@style/TextAppearance.AppCompat.Large" />

        <com.google.android.material.textfield.TextInputLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            android:hint="비밀번호"
            app:endIconMode="password_toggle"
            app:startIconDrawable="@drawable/key_24px">

            <com.google.android.material.textfield.TextInputEditText
```

```
                <com.google.android.material.textfield.TextInputEditText
                    android:id="@+id/textFieldJoinUserPw"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:inputType="text|textPassword"
                    android:textAppearance="@style/TextAppearance.AppCompat.Large" />
            </com.google.android.material.textfield.TextInputLayout>

            <com.google.android.material.textfield.TextInputLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginTop="10dp"
                android:hint="비밀번호 확인"
                app:endIconMode="password_toggle"
                app:startIconDrawable="@drawable/key_24px">

                <com.google.android.material.textfield.TextInputEditText
                    android:id="@+id/textFieldJoinUserPw2"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:inputType="text|textPassword"
                    android:textAppearance="@style/TextAppearance.AppCompat.Large" />
            </com.google.android.material.textfield.TextInputLayout>

            <Button
                android:id="@+id/buttonJoinNext"
                style="@style/Widget.Material3.Button.OutlinedButton"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginTop="10dp"
                android:text="다음"
                android:textAppearance="@style/TextAppearance.AppCompat.Large" />
        </LinearLayout>
    </ScrollView>
</LinearLayout>
```

JoinFragment에 기본 코드를 작성해준다.

```
[JoinFragment.kt]

    lateinit var fragmentJoinBinding: FragmentJoinBinding
    lateinit var mainActivity: MainActivity

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {
        // Inflate the layout for this fragment

        fragmentJoinBinding = FragmentJoinBinding.inflate(inflater)
        mainActivity = activity as MainActivity

        return fragmentJoinBinding.root
    }
```

툴바를 설정하는 메서드를 만들어준다. 타이틀을 설정하고 이전 화면으로 가기 위한 네비게이션을 구현해준다.

```
[JoinFragment.kt]

    // 툴바 설정
    fun settingToolbar(){
        fragmentJoinBinding.apply {
            toolbarJoin.apply {
                // 타이틀
                title = "회원가입"
                // Back
                setNavigationIcon(R.drawable.arrow_back_24px)
                setNavigationOnClickListener {
                    // 이전 화면으로 간다.
                    mainActivity.removeFragment(MainFragmentName.JOIN_FRAGMENT)
                }
            }
        }
    }
```

메서드를 호출한다.

```
[JoinFragment.kt - onCreateView]
        settingToolbar()
```

007 AddUserInfoFragment 작업

AddUserInfoFragment 를 생성한다.

Fragment 이름을 등록해준다.

```
[Tools.kt]

// MainActivity에서 보여줄 프래그먼트들의 이름
enum class MainFragmentName(var str:String){
    LOGIN_FRAGMENT("LoginFragment"),
    JOIN_FRAGMENT("JoinFragment"),
    ADD_USER_INFO_FRAGMENT("AddUserInfoFragment"),
}
```

MainActivity에 Fragment 객체 생성하는 코드를 넣어준다.

```
[MainActivity.kt - replaceFragment]
            // 회원가입 화면 2
            MainFragmentName.ADD_USER_INFO_FRAGMENT -> {
                newFragment = AddUserInfoFragment()
            }
```

JoinFragment 에 다음 버튼을 눌렀을 때 AddUserInfoFragment가 보여지도록 한다.

```
[JoinFragment.kt]

    // 다음 버튼
    fun settingButtonJoinNext(){
        fragmentJoinBinding.apply {
            buttonJoinNext.apply {
                // 버튼을 눌렀을 때
                setOnClickListener {
                    // AddUserInfoFragment를 보여준다.
                    mainActivity.replaceFragment(MainFragmentName.ADD_USER_INFO_FRAGMENT, true, true, null)
                }
            }
        }
    }
```

메서드를 호출한다.

```
[JoinFragment.kt - conCreateView]

        settingButtonJoinNext()
```

화면을 구성해준다.

```
[fragemtn_add_user_info.xml]

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:transitionGroup="true"
    tools:context=".fragment.AddUserInfoFragment" >

    <com.google.android.material.appbar.MaterialToolbar
        android:id="@+id/toolbarAddUserInfo"
        style="@style/Theme.AndroidProject4BoardApp.Toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:minHeight="?attr/actionBarSize"
        android:theme="?attr/actionBarTheme" />

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical"
            android:padding="10dp" >

            <com.google.android.material.textfield.TextInputLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:hint="닉네임"
                app:endIconMode="clear_text"
                app:startIconDrawable="@drawable/person_add_24px">

                <com.google.android.material.textfield.TextInputEditText
                    android:id="@+id/textFieldAddUserInfoNickName"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:inputType="text"
                    android:textAppearance="@style/TextAppearance.AppCompat.Large" />
            </com.google.android.material.textfield.TextInputLayout>

            <com.google.android.material.textfield.TextInputLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginTop="10dp"
                android:hint="나이"
                app:endIconMode="clear_text"
                app:startIconDrawable="@drawable/face_24px">

                <com.google.android.material.textfield.TextInputEditText
                    android:id="@+id/textFieldAddUserInfoAge"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:inputType="number|numberDecimal"
                    android:textAppearance="@style/TextAppearance.AppCompat.Large" />
            </com.google.android.material.textfield.TextInputLayout>
```

```
<com.google.android.material.button.MaterialButtonToggleGroup
    android:id="@+id/toggleAddUserInfoGender"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp"
    app:selectionRequired="true"
    app:singleSelection="true">

    <Button
        android:id="@+id/buttonAddUserInfoMale"
        style="@style/Widget.Material3.Button.OutlinedButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="남자"
        android:textAppearance="@style/TextAppearance.AppCompat.Large" />

    <Button
        android:id="@+id/buttonAddUserInfoFemale"
        style="@style/Widget.Material3.Button.OutlinedButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="여자"
        android:textAppearance="@style/TextAppearance.AppCompat.Large" />

</com.google.android.material.button.MaterialButtonToggleGroup>

<com.google.android.material.divider.MaterialDivider
    android:layout_width="match_parent"
    android:layout_height="2dp"
    android:layout_marginTop="10dp"
    android:background="@color/black" />

<com.google.android.material.checkbox.MaterialCheckBox
    android:id="@+id/checkBoxAddUserInfoAll"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp"
    android:text="취미"
    android:textAppearance="@style/TextAppearance.AppCompat.Large" />

<LinearLayout
    android:id="@+id/checkBoxGroupAddUserInfo1"
```

```
        android:id="@+id/checkBoxGroupAddUserInfo1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginStart="10dp"
        android:layout_marginTop="10dp"
        android:orientation="horizontal">

        <com.google.android.material.checkbox.MaterialCheckBox
            android:id="@+id/checkBoxAddUserInfoHobby1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="운동" />

        <com.google.android.material.checkbox.MaterialCheckBox
            android:id="@+id/checkBoxAddUserInfoHobby2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="독서" />

        <com.google.android.material.checkbox.MaterialCheckBox
            android:id="@+id/checkBoxAddUserInfoHobby3"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="영화감상" />

    </LinearLayout>

    <LinearLayout
        android:id="@+id/checkBoxGroupAddUserInfo2"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginStart="10dp"
        android:layout_marginTop="10dp"
        android:orientation="horizontal">

        <com.google.android.material.checkbox.MaterialCheckBox
            android:id="@+id/checkBoxAddUserInfoHobby4"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="요리" />
```

```
                <com.google.android.material.checkbox.MaterialCheckBox
                    android:id="@+id/checkBoxAddUserInfoHobby5"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:layout_weight="1"
                    android:text="음악" />

                <com.google.android.material.checkbox.MaterialCheckBox
                    android:id="@+id/checkBoxAddUserInfoHobby6"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:layout_weight="1"
                    android:text="기타" />

            </LinearLayout>

            <Button
                android:id="@+id/buttonAddUserInfoSubmit"
                style="@style/Widget.Material3.Button.OutlinedButton"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginTop="10dp"
                android:text="가입"
                android:textAppearance="@style/TextAppearance.AppCompat.Large" />
        </LinearLayout>
    </ScrollView>
</LinearLayout>
```

AddUserInfoFragment 에 기본 코드를 작성해준다.

```
    lateinit var fragmentAddUserInfoBinding: FragmentAddUserInfoBinding
    lateinit var mainActivity: MainActivity

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {
        // Inflate the layout for this fragment

        fragmentAddUserInfoBinding = FragmentAddUserInfoBinding.inflate(inflater)
        mainActivity = activity as MainActivity

        return fragmentAddUserInfoBinding.root
    }
```

툴바 설정 메서드를 구현해준다.

```
[AddUserInfoFragment.kt]

    // 툴바 설정
    fun settingToolbar(){
        fragmentAddUserInfoBinding.apply {
            toolbarAddUserInfo.apply {
                // 타이틀
                title = "회원가입"
                // Back
                setNavigationIcon(R.drawable.arrow_back_24px)
                setNavigationOnClickListener {
                    mainActivity.removeFragment(MainFragmentName.ADD_USER_INFO_FRAGMENT)
                }
            }
        }
    }
```

메서드를 호출해준다.

```
settingToolbar()
```

가입 완료 버튼을 구현해준다.

```
    // 가입 버튼
    fun settingButtonAddUserInfoSubmit(){
        fragmentAddUserInfoBinding.apply {
            buttonAddUserInfoSubmit.apply {
                // 눌렀을 때
                setOnClickListener {
                    mainActivity.removeFragment(MainFragmentName.ADD_USER_INFO_FRAGMENT)
                    mainActivity.removeFragment(MainFragmentName.JOIN_FRAGMENT)
                }
            }
        }
    }
```

메서드를 호출해준다.

```
        settingButtonAddUserInfoSubmit()
```

008 ContentActivity 작업

ContentActivity 를 생성한다.

LoginFragment 에 로그인 버튼 설정 메서드를 만들어준다.

```
[LoginFragment.kt]

    // 로그인 버튼
    fun settingButtonLoginSubmit(){
        fragmentLoginBinding.apply {
            buttonLoginSubmit.apply {
                // 버튼을 눌렀을 때
                setOnClickListener {
                    // ContentActivity를 실행한다.
                    val contentIntent = Intent(mainActivity, ContentActivity::class.java)
                    startActivity(contentIntent)
                    // MainActivity를 종료한다.
                    mainActivity.finish()
                }
            }
        }
    }
```

메서드를 호출해준다.

```
[LoginFragment.kt - onCreateView]

        settingButtonLoginSubmit()
```

AndroidManifest.xml 에서 ContentActivity 의 테마를 설정해준다.

```
[ AndroidManifest.xml ]

        <activity
            android:name=".ContentActivity"
            android:exported="false"
            android:theme="@style/Base.Theme.AndroidProject4BoardApp"/>
```

ContentActivity의 화면을 구성해준다.

```
[ activity_content.xml ]

<?xml version="1.0" encoding="utf-8"?>
<androidx.drawerlayout.widget.DrawerLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/drawerLayoutContent"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ContentActivity">

    <androidx.coordinatorlayout.widget.CoordinatorLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="vertical">

            <Button
                android:id="@+id/buttonTest"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="Button" />
        </LinearLayout>
    </androidx.coordinatorlayout.widget.CoordinatorLayout>

    <com.google.android.material.navigation.NavigationView
        android:id="@+id/navigationViewContent"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        app:menu="@menu/menu_content_drawer" />
</androidx.drawerlayout.widget.DrawerLayout>
```

ContentActivity에 기본 코드를 넣어준다.

```
[ ContentActivity.kt ]

    lateinit var activityContentBinding: ActivityContentBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        activityContentBinding = ActivityContentBinding.inflate(layoutInflater)
        setContentView(activityContentBinding.root)
    }
```

버튼을 누르면 NavigationDrawer를 오픈 시킨다.

```
[ ContentActivity.kt - onCreate ]

        activityContentBinding.buttonTest.setOnClickListener {
            activityContentBinding.drawerLayoutContent.open()
        }
```

NavigationView 상단에 보여줄 헤더를 구성하기 위한 레이아웃 파일을 만들어준다.

```
[res/layout/header_content_drawer.xml]

<?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"
    android:orientation="vertical"
    android:padding="10dp">

    <TextView
        android:id="@+id/headerContentDrawerNickName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="TextView"
        android:textAppearance="@style/TextAppearance.AppCompat.Large" />

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:text="환영합니다" />
</LinearLayout>
```

네비게이션 뷰를 설정하는 메서드를 만들어준다.

```
[ContentActivity.kt]

    // 네비게이션 뷰 설정
    fun settingNavigationView(){
        activityContentBinding.apply {
            navigationViewContent.apply {
                // 헤더로 보여줄 View를 생성한다.
                val headerContentDrawerBinding = HeaderContentDrawerBinding.inflate(layoutInflater)
                // 헤더로 보여줄 View를 설정한다.
                addHeaderView(headerContentDrawerBinding.root)

                // 사용자 닉네임을 설정한다.
                headerContentDrawerBinding.headerContentDrawerNickName.text = "홍길동님"
            }
        }
    }
```

메서드를 호출해준다.

```
[ ContentActivity.kt - onCreate]

        settingNavigationView()
```

메뉴 파일을 구성해준다.

```
[res/menu/menu_content_drawer.xml]

<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:title="게시판" >
        <menu >
            <item
                android:id="@+id/menuItemContentNavigationAll"
                android:checkable="true"
                android:icon="@drawable/local_shipping_24px"
                android:title="전체게시판"
                app:showAsAction="ifRoom" />
```

```
            app:showAsAction="ifRoom" />
        <item
            android:id="@+id/menuItemContentNavigation1"
            android:checkable="true"
            android:icon="@drawable/post_add_24px"
            android:title="자유게시판"
            app:showAsAction="ifRoom" />
        <item
            android:id="@+id/menuItemContentNavigation2"
            android:checkable="true"
            android:icon="@drawable/compost_24px"
            android:title="유머게시판"
            app:showAsAction="ifRoom" />
        <item
            android:id="@+id/menuItemContentNavigation3"
            android:checkable="true"
            android:icon="@drawable/package_24px"
            android:title="시사게시판"
            app:showAsAction="ifRoom" />
        <item
            android:id="@+id/menuItemContentNavigation4"
            android:checkable="true"
            android:icon="@drawable/approval_24px"
            android:title="스포치게시판"
            app:showAsAction="ifRoom" />
    </menu>
</item>
<item
    android:id="@+id/menuItemContentNavigationModifyUserInfo"
    android:checkable="true"
    android:icon="@drawable/person_24px"
    android:title="사용자 정보 수정"
    app:showAsAction="ifRoom" />
<item
    android:id="@+id/menuItemContentNavigationLogout"
    android:checkable="true"
    android:icon="@drawable/logout_24px"
    android:title="로그아웃"
    app:showAsAction="ifRoom" />
<item
    android:id="@+id/menuItemContentNavigationSignOut"
    android:checkable="true"
    android:icon="@drawable/move_item_24px"
    android:title="회원탈퇴" />
</menu>
```

```
</menu>
```

네비게이션의 메뉴를 누르면 네비게이션 뷰가 닫히도록 한다.

```
[ ContentActivity - settingNavigationView]

                // 메뉴를 눌렀을 때 동작하는 리스너
                setNavigationItemSelectedListener {

                    // 딜레이
                    SystemClock.sleep(200)

                    // 메뉴의 id로 분기한다.
                    when(it.itemId){
                        // 전체 게시판
                        R.id.menuItemContentNavigationAll -> {
                            // NavigationView를 닫아준다.
                            drawerLayoutContent.close()
                        }
                        // 자유 게시판
                        R.id.menuItemContentNavigation1 -> {
                            // NavigationView를 닫아준다.
                            drawerLayoutContent.close()
                        }
                    }

                    true
                }
```

activity_content.xml 에서 버튼을 삭제한다.

버튼을 제거한 자리에 FragmentContainerView를 배치한다.

```
[ activity_content.xml ]

            <androidx.fragment.app.FragmentContainerView
                android:id="@+id/containerContent"
                android:layout_width="match_parent"
                android:layout_height="wrap_content" />
```

ContentActivity에서 버튼 관련 코드를 삭제한다.

```
[ ContentActivity.kt - onCreate]

아래의 코드를 삭제한다.

        activityContentBinding.buttonTest.setOnClickListener {
            activityContentBinding.drawerLayoutContent.open()
        }
```

009 Fragment 초기 작업

Tools.kt 에 프래그먼트 이름을 관리할 enum class 를 작성한다.

```
[Tools.kt]

// ContentActivity에서 보여줄 프래그먼트들의 이름
enum class ContentFragmentName(var str:String){
    A_FRAGMENT("a"),
    B_FRAGMENT("b"),
}
```

ContentActivity 상단에 프래그먼트를 담을 프로퍼티 정의

```
[ContentActivity.kt]

    // 프래그먼트 객체를 담을 변수
    var oldFragment:Fragment? = null
    var newFragment:Fragment? = null
```

fragment basic code 를 복사에 넣어준다.

replaceFragment 메서드의 첫 번째 매개변수의 타입을 변경한다.

```
fun replaceFragment(name:ContentFragmentName, addToBackStack:Boolean, isAnimate:Boolean, data:Bundle?){
```

Fragment를 설정해줄 FragmentContainerView의 id 를 변경한다.

```
fragmentTransaction.replace(R.id.containerContent, newFragment!!)
```

removeFragment 메서드의 첫 번째 매개변수의 타입을 변경한다.

```
fun removeFragment(name:ContentFragmentName){
```

import 를 해준다.

010 MainFragment 작업

MainFragment 를 생성한다.

Fragment의 이름을 등록해준다.

```
[Tools.kt]

// ContentActivity에서 보여줄 프래그먼트들의 이름
enum class ContentFragmentName(var str:String){
    MAIN_FRAGMENT("MainFragment"),
    B_FRAGMENT("b"),
}
```

Fragment 객체 생성하는 코드를 넣어준다.

```
[ContentActivity.kt - replaceFragment]

        when(name){
            // 게시글 목록 화면
            ContentFragmentName.MAIN_FRAGMENT -> {
                newFragment = MainFragment()
            }
            ContentFragmentName.B_FRAGMENT -> {

            }
        }
```

MainFragment가 나타나도록 한다.

```
[ContentActivity.kt - onCreateView]

        // MainFragment가 나타나도록 한다.
        replaceFragment(ContentFragmentName.MAIN_FRAGMENT, false, false, null)
```

fragment_main.xml 의 화면을 구성해준다.

중요!!!! SearchView의 layout_anchor 속성에 SearchBar의 id를 넣어 서로 연결되게 해야 한다.

```
[res/layout/fragment_main.xml]

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:transitionGroup="true"
    tools:context=".fragment.MainFragment" >

    <com.google.android.material.appbar.MaterialToolbar
        android:id="@+id/toolbarMain"
        style="@style/Theme.AndroidProject4BoardApp.Toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:minHeight="?attr/actionBarSize"
```

```
        android:minHeight="?attr/actionBarSize"
        android:theme="?attr/actionBarTheme" />

    <androidx.coordinatorlayout.widget.CoordinatorLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="10dp">

        <com.google.android.material.search.SearchBar
            android:id="@+id/searchBarMain"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />

        <androidx.recyclerview.widget.RecyclerView
            android:id="@+id/recyclerViewMain"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_marginTop="90dp" />

        <com.google.android.material.search.SearchView
            android:id="@+id/searchViewMain"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            app:layout_anchor="@id/searchBarMain">

            <androidx.recyclerview.widget.RecyclerView
                android:id="@+id/recyclerViewMainSearch"
                android:layout_width="match_parent"
                android:layout_height="match_parent" />
        </com.google.android.material.search.SearchView>
    </androidx.coordinatorlayout.widget.CoordinatorLayout>
</LinearLayout>
```

MainFragment에 기본 코드를 작성한다.

```
    lateinit var fragmentMainBinding: FragmentMainBinding
    lateinit var contentActivity: ContentActivity

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {
        // Inflate the layout for this fragment

        fragmentMainBinding = FragmentMainBinding.inflate(inflater)
        contentActivity = activity as ContentActivity

        return fragmentMainBinding.root
    }
```

툴바를 설정하는 메서드를 구현한다.

```
[MainFragment.kt]

    // 툴바 설정
    fun settingToolbar(){
        fragmentMainBinding.apply {
            toolbarMain.apply {
                // 타이틀
                title = "전체 게시판"
                // 네비게이션
                setNavigationIcon(R.drawable.menu_24px)
                setNavigationOnClickListener {
                    // Drawer 메뉴가 나타나게 한다.
                    contentActivity.activityContentBinding.drawerLayoutContent.open()
                }
            }
        }
    }
```

메서드를 호출한다.

```
[MainFragment - onCreateView]

        settingToolbar()
```

RecyclerView의 항목을 구성하기 위한 xml 파일을 작성해준다.

```
[res/layout/row_main.xml]

<?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"
    android:orientation="vertical"
    android:padding="10dp">

    <TextView
        android:id="@+id/textViewRowMainSubject"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="TextView"
        android:textAppearance="@style/TextAppearance.AppCompat.Large" />

    <TextView
        android:id="@+id/textViewRowMainNickName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:text="TextView" />
</LinearLayout>
```

메인 화면의 RecyclerView의 Adapter를 작성해준다.

[MainFragment.kt]

```
    // 메인 화면의 RecyclerView의 어뎁터
    inner class MainRecyclerViewAdapter : RecyclerView.Adapter<MainRecyclerViewAdapter.MainViewHolder>(){
        inner class MainViewHolder(rowMainBinding: RowMainBinding) : RecyclerView.ViewHolder(rowMainBinding.root){
            val rowMainBinding:RowMainBinding

            init {
                this.rowMainBinding = rowMainBinding

                this.rowMainBinding.root.layoutParams = ViewGroup.LayoutParams(
                    ViewGroup.LayoutParams.MATCH_PARENT,
                    ViewGroup.LayoutParams.WRAP_CONTENT
                )
            }
        }

        override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MainViewHolder {
            val rowMainBinding = RowMainBinding.inflate(layoutInflater)
            val mainViewHolder = MainViewHolder(rowMainBinding)
            return mainViewHolder
        }

        override fun getItemCount(): Int {
            return 100
        }

        override fun onBindViewHolder(holder: MainViewHolder, position: Int) {
            holder.rowMainBinding.textViewRowMainSubject.text = "제목 $position"
            holder.rowMainBinding.textViewRowMainNickName.text = "작성자 $position"
        }
    }
```

검색화면의 RecyclerView의 어뎁터를 작성해준다.

```
    // 검색 화면의 RecyclerView의 어뎁터
    inner class SearchRecyclerViewAdapter : RecyclerView.Adapter<SearchRecyclerViewAdapter.SearchViewHolder>(){
        inner class SearchViewHolder(rowMainBinding: RowMainBinding) : RecyclerView.ViewHolder(rowMainBinding.root){
            val rowMainBinding:RowMainBinding

            init {
                this.rowMainBinding = rowMainBinding

                this.rowMainBinding.root.layoutParams = ViewGroup.LayoutParams(
                    ViewGroup.LayoutParams.MATCH_PARENT,
                    ViewGroup.LayoutParams.WRAP_CONTENT
                )
            }
        }

        override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): SearchViewHolder {
            val rowMainBinding = RowMainBinding.inflate(layoutInflater)
            val searchViewHolder = SearchViewHolder(rowMainBinding)
            return searchViewHolder
        }

        override fun getItemCount(): Int {
            return 100
        }

        override fun onBindViewHolder(holder: SearchViewHolder, position: Int) {
            holder.rowMainBinding.textViewRowMainSubject.text = "제목 $position"
            holder.rowMainBinding.textViewRowMainNickName.text = "작성자 $position"
        }
    }
```

메인 화면의 RecyclerView를 설정하는 메서드를 만들어준다.

```
[MainFragment.kt]

    // 메인 화면의 RecyclerView 설정
    fun settingRecyclerViewMain(){
        fragmentMainBinding.apply {
            recyclerViewMain.apply {
                // 어뎁터
                adapter = MainRecyclerViewAdapter()
                // 레이아웃 매니저
                layoutManager = LinearLayoutManager(contentActivity)
                // 데코레이션
                val deco = MaterialDividerItemDecoration(contentActivity, MaterialDividerItemDecoration.VERTICAL)
                addItemDecoration(deco)
            }
        }
    }
```

메서드를 호출한다.

```
[MainFragment.kt - conCreateView ]

        settingRecyclerViewMain()
```

검색 화면의 RecyclerView를 설정하는 메서드를 만들어준다

```
[MainFragment.kt]

    // 검색 화면의 RecyclerView를 구성하는 메서드
    fun settingRecyclerViewMainSearch(){
        fragmentMainBinding.apply {
            recyclerViewMainSearch.apply {
                // 어뎁터
                adapter = SearchRecyclerViewAdapter()
                // 레이아웃 매니저
                layoutManager = LinearLayoutManager(contentActivity)
                // 데코레이션
                val deco = MaterialDividerItemDecoration(contentActivity, MaterialDividerItemDecoration.VERTICAL)
                addItemDecoration(deco)
            }
        }
    }
```

메서드를 호출한다.

```
[MainFragment.kt - onCreateView]

        settingRecyclerViewMainSearch()
```

SearchView에 배치할 메뉴를 만들기 위한 xml 파일을 작성한다

```
[ res/menu/menu_main_search_menu.xml]

<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:id="@+id/menuItemMainSearchAdd"
        android:icon="@drawable/add_24px"
        android:title="글작성"
        app:showAsAction="always" />
</menu>
```

메서드를 호출해준다.

```
[ MainFragment.kt - onCreateView]

        settingSearchBar()
```

011 AddContentFragment 작업

AddContentFragment 를 생성해준다.

Fragment 이름을 등록해준다

```
[Tools.kt]

// ContentActivity에서 보여줄 프래그먼트들의 이름
enum class ContentFragmentName(var str:String){
    MAIN_FRAGMENT("MainFragment"),
    ADD_CONTENT_FRAGMENT("AddContentFragment"),
}
```

Fragment 생성하는 코드를 작성해준다.

```
[ ContentActivity.kt ]

        // 이름으로 분기한다.
        // Fragment의 객체를 생성하여 변수에 담아준다.
        when(name){
            // 게시글 목록 화면
            ContentFragmentName.MAIN_FRAGMENT -> {
                newFragment = MainFragment()
            }
            // 게시글 작성 화면
            ContentFragmentName.ADD_CONTENT_FRAGMENT -> {
                newFragment = AddContentFragment()
            }
        }
```

MainFragment 에서 + 메뉴를 누르면 글 작성화면이 나오도록 한다.

```
[ MainFragment - settingSearchBar]

                setOnMenuItemClickListener {
                    // 글 작성 화면이 나타나게 한다.
                    contentActivity.replaceFragment(ContentFragmentName.ADD_CONTENT_FRAGMENT, true, true, null)
                    true
                }
```

화면을 구성해준다

```
[fragment_add_content.xml]

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:transitionGroup="true"
    tools:context=".fragment.AddContentFragment" >

    <com.google.android.material.appbar.MaterialToolbar
        android:id="@+id/toolbarAddContent"
        style="@style/Theme.AndroidProject4BoardApp.Toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:minHeight="?attr/actionBarSize"
        android:theme="?attr/actionBarTheme" />

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical"
            android:padding="10dp" >

            <com.google.android.material.textfield.TextInputLayout
                android:layout_width="match_parent"
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="제목"
        app:endIconMode="clear_text"
        app:startIconDrawable="@drawable/subject_24px">

        <com.google.android.material.textfield.TextInputEditText
            android:id="@+id/textFieldAddContentSubject"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:inputType="text"
            android:textAppearance="@style/TextAppearance.AppCompat.Large" />
    </com.google.android.material.textfield.TextInputLayout>

    <com.google.android.material.button.MaterialButtonToggleGroup
        android:id="@+id/toggleAddContentType"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        app:selectionRequired="true"
        app:singleSelection="true">

        <Button
            android:id="@+id/buttonAddContentType1"
            style="@style/Widget.Material3.Button.OutlinedButton"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="자유"
            android:textAppearance="@style/TextAppearance.AppCompat.Medium" />

        <Button
            android:id="@+id/buttonAddContentType2"
            style="@style/Widget.Material3.Button.OutlinedButton"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="유머"
            android:textAppearance="@style/TextAppearance.AppCompat.Medium" />

        <Button
            android:id="@+id/buttonAddContentType3"
            style="@style/Widget.Material3.Button.OutlinedButton"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
```

```
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="시사"
                android:textAppearance="@style/TextAppearance.AppCompat.Medium" />

            <Button
                android:id="@+id/buttonAddContentType4"
                style="@style/Widget.Material3.Button.OutlinedButton"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="스포츠"
                android:textAppearance="@style/TextAppearance.AppCompat.Medium" />
        </com.google.android.material.button.MaterialButtonToggleGroup>

        <com.google.android.material.textfield.TextInputLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            android:hint="내용"
            app:endIconMode="clear_text"
            app:startIconDrawable="@drawable/description_24px">

            <com.google.android.material.textfield.TextInputEditText
                android:id="@+id/textFieldAddContentText"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:inputType="text|textMultiLine"
                android:textAppearance="@style/TextAppearance.AppCompat.Large" />
        </com.google.android.material.textfield.TextInputLayout>

        <ImageView
            android:id="@+id/imageViewAddContent"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            android:adjustViewBounds="true"
            app:srcCompat="@drawable/panorama_24px" />

    </LinearLayout>
  </ScrollView>
</LinearLayout>
```

사용할 메뉴 파일을 만들어준다.

```
[res/menu/menu_add_content.xml]

<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:id="@+id/menuItemAddContentCamera"
        android:icon="@drawable/photo_camera_24px"
        android:title="카메라"
        app:showAsAction="always" />
    <item
        android:id="@+id/menuItemAddContentAlbum"
        android:icon="@drawable/photo_album_24px"
        android:title="앨범"
        app:showAsAction="always" />
    <item
        android:id="@+id/menuItemAddContentReset"
        android:icon="@drawable/clear_all_24px"
        android:title="초기화"
        app:showAsAction="always" />
    <item
        android:id="@+id/menuItemAddContentDone"
        android:icon="@drawable/done_24px"
        android:title="완료"
        app:showAsAction="always" />
</menu>
```

AddContentFragment 에 기본 코드를 작성한다.

[AddContentFragment.kt]

```
    lateinit var fragmentAddContentBinding: FragmentAddContentBinding
    lateinit var contentActivity: ContentActivity

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {
        // Inflate the layout for this fragment

        fragmentAddContentBinding = FragmentAddContentBinding.inflate(inflater)
        contentActivity = activity as ContentActivity

        return fragmentAddContentBinding.root
    }
```

툴바를 설정하는 메서드를 만들어준다.

[AddContentFragment.kt]

```
    // 툴바 셋팅
    fun settingToolbarAddContent(){
        fragmentAddContentBinding.apply {
            toolbarAddContent.apply {
                // 타이틀
                title = "글 작성"
                // Back
                setNavigationIcon(R.drawable.arrow_back_24px)
                setNavigationOnClickListener {
                    contentActivity.removeFragment(ContentFragmentName.ADD_CONTENT_FRAGMENT)
                }
                // 메뉴
                inflateMenu(R.menu.menu_add_content)
            }
        }
    }
```

메서드를 호출해준다.

```
        settingToolbarAddContent()
```

012 ReadContentFragment 작업

ReadContentFragment 를 생성한다.

Tools.kt 에 프래그먼트 이름을 등록한다.

```
[Tools.kt]

// ContentActivity에서 보여줄 프래그먼트들의 이름
enum class ContentFragmentName(var str:String){
    MAIN_FRAGMENT("MainFragment"),
    ADD_CONTENT_FRAGMENT("AddContentFragment"),
    READ_CONTENT_FRAGMENT("ReadContentFragment"),
}
```

Fragment 객체 생성하는 코드를 작성해준다.

```
[ContentFragment.kt - replaceFragment]

            // 게시글 읽기 화면
            ContentFragmentName.READ_CONTENT_FRAGMENT -> {
                newFragment = ReadContentFragment()
            }
```

AddContentFragment에 네비게이션 코드를 작성해준다.

```
[AddContentFragment.kt - settingToolbarAddContent ]

                setOnMenuItemClickListener {
                    // 메뉴의 id로 분기한다.
                    when(it.itemId){
                        // 카메라
                        R.id.menuItemAddContentCamera -> {

                        }
                        // 앨범
                        R.id.menuItemAddContentAlbum -> {

                        }
                        // 초기화
                        R.id.menuItemAddContentReset -> {

                        }
                        // 완료
                        R.id.menuItemAddContentDone -> {
                            // ReadContentFragment로 이동한다.

                            contentActivity.replaceFragment(ContentFragmentName.READ_CONTENT_FRAGMENT, true, true, null)
                        }
                    }
                    true
                }
```

화면을 구섲한다.

```
[fragment_read_content.xml]

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:transitionGroup="true"
    tools:context=".fragment.ReadContentFragment" >

    <com.google.android.material.appbar.MaterialToolbar
```

```
<com.google.android.material.appbar.MaterialToolbar
    android:id="@+id/toolbarReadContent"
    style="@style/Theme.AndroidProject4BoardApp.Toolbar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:minHeight="?attr/actionBarSize"
    android:theme="?attr/actionBarTheme" />

<ScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:padding="10dp" >

        <com.google.android.material.textfield.TextInputLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="제목"
            app:startIconDrawable="@drawable/subject_24px">

            <com.google.android.material.textfield.TextInputEditText
                android:id="@+id/textFieldReadContentSubject"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:clickable="false"
                android:cursorVisible="false"
                android:focusable="false"
                android:inputType="text"
                android:textAppearance="@style/TextAppearance.AppCompat.Large" />
        </com.google.android.material.textfield.TextInputLayout>

        <com.google.android.material.textfield.TextInputLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            android:hint="게시판 종류"
            app:startIconDrawable="@drawable/warning_24px">

            <com.google.android.material.textfield.TextInputEditText
                android:id="@+id/textFieldReadContentType"
                android:layout_width="match_parent"
```

```
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:clickable="false"
            android:cursorVisible="false"
            android:focusable="false"
            android:inputType="text"
            android:textAppearance="@style/TextAppearance.AppCompat.Large" />
    </com.google.android.material.textfield.TextInputLayout>

    <com.google.android.material.textfield.TextInputLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:hint="닉네임"
        app:startIconDrawable="@drawable/person_add_24px">

        <com.google.android.material.textfield.TextInputEditText
            android:id="@+id/textFieldReadContentNickName"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:clickable="false"
            android:cursorVisible="false"
            android:focusable="false"
            android:inputType="text"
            android:textAppearance="@style/TextAppearance.AppCompat.Large" />
    </com.google.android.material.textfield.TextInputLayout>

    <com.google.android.material.textfield.TextInputLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:hint="작성 날짜"
        app:startIconDrawable="@drawable/calendar_month_24px">

        <com.google.android.material.textfield.TextInputEditText
            android:id="@+id/textFieldReadContentDate"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:clickable="false"
            android:cursorVisible="false"
            android:focusable="false"
            android:inputType="text"
            android:textAppearance="@style/TextAppearance.AppCompat.Large" />
    </com.google.android.material.textfield.TextInputLayout>
```

```
            <com.google.android.material.textfield.TextInputLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginTop="10dp"
                android:hint="글 내용"
                app:startIconDrawable="@drawable/description_24px">

                <com.google.android.material.textfield.TextInputEditText
                    android:id="@+id/textFieldReadContentText"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:clickable="false"
                    android:cursorVisible="false"
                    android:focusable="false"
                    android:inputType="text|textMultiLine"
                    android:textAppearance="@style/TextAppearance.AppCompat.Large" />
            </com.google.android.material.textfield.TextInputLayout>

            <ImageView
                android:id="@+id/imageViewReadContent"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginTop="10dp"
                android:adjustViewBounds="true"
                app:srcCompat="@drawable/panorama_24px" />
        </LinearLayout>
    </ScrollView>
</LinearLayout>
```

ReadContentFragment 에 기본 코드를 작성해준다.

```
[ReadCcontentFragment.kt]

    lateinit var fragmentReadContentBinding: FragmentReadContentBinding
    lateinit var contentActivity: ContentActivity

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {
        // Inflate the layout for this fragment

        fragmentReadContentBinding = FragmentReadContentBinding.inflate(inflater)
        contentActivity = activity as ContentActivity

        return fragmentReadContentBinding.root
    }
```

뒤로가기 처리 메서드를 만들어준다.

```
[ReadContentFragment.kt]

    // 뒤로가기 처리
    fun backProcesss(){
        contentActivity.removeFragment(ContentFragmentName.READ_CONTENT_FRAGMENT)
        contentActivity.removeFragment(ContentFragmentName.ADD_CONTENT_FRAGMENT)
    }
```

툴바를 설정하는 메서드를 만들어준다.

```
[ ReadContentFragment.kt ]

    // 툴바 설정
    fun settingToolbarReadContent(){
        fragmentReadContentBinding.apply {
            toolbarReadContent.apply {
                // 타이틀
                title = "글 읽기"
                // 네비게이션
                setNavigationIcon(R.drawable.arrow_back_24px)
                setNavigationOnClickListener {
                    backProcesss()
                }
            }
        }
    }
```

메서드를 호출한다

```
[ReadContentFragment.kt - onCreateView]

settingToolbarReadContent()
```

백 버튼을 눌렀을 때 처리 메서드를 구현한다.

```
[ReadContentFragment.kt]

    // Back button 눌렀을 때
    fun settingBackButton(){
        contentActivity.onBackPressedDispatcher.addCallback {
            // 뒤로가기 처리 메서드 호출
            backProcesss()
            // 뒤로가기 버튼의 콜백을 제거한다.
            remove()
        }
    }
```

메서드를 호출한다.

```
[ReadContentFragment.kt - onCreateView]

        settingBackButton()
```

메뉴파일을 작성해준다.

```
[res/menu/menu_read_content.xml]

<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:id="@+id/menuItemReadContentReply"
        android:icon="@drawable/description_24px"
        android:title="댓글"
        app:showAsAction="always" />
    <item
        android:id="@+id/menuItemReadContentModify"
        android:icon="@drawable/edit_24px"
        android:title="수정하기"
        app:showAsAction="always" />
    <item
        android:id="@+id/menuItemReadContentDelete"
        android:icon="@drawable/delete_24px"
        android:title="삭제하기"
        app:showAsAction="always" />
</menu>
```

ReadContentFragment 에 메뉴를 구성해준다.

[ReadContentFragment.kt - settingToolbarReadContent]

```
                // 메뉴
                inflateMenu(R.menu.menu_read_content)
                setOnMenuItemClickListener {
                    // 메뉴의 id로 분기한다.
                    when(it.itemId){
                        // 댓글
                        R.id.menuItemReadContentReply -> {

                        }
                        // 수정하기
                        R.id.menuItemReadContentModify -> {

                        }
                        // 삭제하기
                        R.id.menuItemReadContentDelete -> {

                        }
                    }
                    true
                }
```

ReadContentBottomFragment를 생성하고 기본 코드를 작성한다. 부모 클래스는 BottomSheetDialogFragment로 변경한다.

```
[ReadContentBottomFragment.kt]

class ReadContentBottomFragment : BottomSheetDialogFragment() {

    lateinit var fragmentReadContentBottomBinding: FragmentReadContentBottomBinding
    lateinit var contentActivity: ContentActivity

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {
        // Inflate the layout for this fragment

        fragmentReadContentBottomBinding = FragmentReadContentBottomBinding.inflate(inflater)
        contentActivity = activity as ContentActivity

        return fragmentReadContentBottomBinding.root
    }

}
```

BottomSheet를 띄우는 메서드를 만들어준다.

```
[ReadContentFragment.kt]

    // 댓글을 보여불 BottomSheet를 띄워준다.
    fun showReplyBottomSheet(){
        val readContentBottomFragment = ReadContentBottomFragment()
        readContentBottomFragment.show(contentActivity.supportFragmentManager, "ReplyBottomSheet")
    }
```

댓글 메뉴를 누르면 BottomSheet가 나오도록 한다.

```
[ReadContentFragment.kt - settingToolbarReadContent]

                        // 댓글
                        R.id.menuItemReadContentReply -> {
                            // 댓글을 보여줄 BottomSheet를 띄운다.
                            showReplyBottomSheet()
                        }
```

화면을 구성해준다.

```
[fragment_read_content_bottom.xml]


<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="20dp"
    tools:context=".fragment.ReadContentBottomFragment" >

    <com.google.android.material.textfield.TextInputLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="10dp"
        android:hint="댓글"
        app:endIconMode="clear_text"
        app:startIconDrawable="@drawable/warning_24px">

        <com.google.android.material.textfield.TextInputEditText
            android:id="@+id/textFieldAddContentReply"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:inputType="text"
            android:textAppearance="@style/TextAppearance.AppCompat.Large" />
    </com.google.android.material.textfield.TextInputLayout>

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerViewAddContentReply"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```

RecyclerView 항목을 위한 화면을 구성해준다.

```
[ res/layout/row_read_content_reply.xml]

<?xml version="1.0" encoding="utf-8"?>
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:padding="10dp" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:orientation="vertical">

        <TextView
            android:id="@+id/textViewRowReplyText"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="TextView"
            android:textAppearance="@style/TextAppearance.AppCompat.Large" />

        <TextView
            android:id="@+id/textViewRowReplyNickName"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            android:text="TextView" />

        <TextView
            android:id="@+id/textViewRowReplyDate"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            android:text="TextView" />
    </LinearLayout>

    <Button
        android:id="@+id/buttonRowReplyDelete"
        style="@style/Widget.Material3.Button.IconButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        app:icon="@drawable/delete_24px" />
</LinearLayout>
```

Recyclerview의 Adapter 클래스를 작성해준다.

```
[ReadContentBottomFragmeht.kt]

    // 댓글 목록을 보여줄 RecyclerView의 어뎁터
    inner class BottomRecyclerViewAdapter : RecyclerView.Adapter<BottomRecyclerViewAdapter.BottomViewHolder>(){

        inner class BottomViewHolder(rowReadContentReplayBinding: RowReadContentReplayBinding) :
RecyclerView.ViewHolder(rowReadContentReplayBinding.root){
            val rowReadContentReplayBinding : RowReadContentReplayBinding

            init {
                this.rowReadContentReplayBinding = rowReadContentReplayBinding

                rowReadContentReplayBinding.root.layoutParams = ViewGroup.LayoutParams(
                    ViewGroup.LayoutParams.MATCH_PARENT,
                    ViewGroup.LayoutParams.WRAP_CONTENT
                )
            }
        }

        override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): BottomViewHolder {
            val rowReadContentReplayBinding = RowReadContentReplayBinding.inflate(layoutInflater)
            val bottomViewHolder = BottomViewHolder(rowReadContentReplayBinding)
            return bottomViewHolder
        }

        override fun getItemCount(): Int {
            return 100
        }

        override fun onBindViewHolder(holder: BottomViewHolder, position: Int) {
            holder.rowReadContentReplayBinding.textViewRowReplyText.text = "댓글입니다 $position"
            holder.rowReadContentReplayBinding.textViewRowReplyNickName.text = "작성자 $position"
            holder.rowReadContentReplayBinding.textViewRowReplyDate.text = "2024-03-07"
        }
    }
```

RecyclerView를 구성하는 메서드를 작성해준다.

```
[ReadContentBottomFragment.kt  ]

    // RecyclerView 구성 메서드
    fun settingRecyclerViewAddContentReply(){
        fragmentReadContentBottomBinding.apply {
            recyclerViewAddContentReply.apply {
                // 어뎁터
                adapter = BottomRecyclerViewAdapter()
                // 레이아웃 매니저
                layoutManager = LinearLayoutManager(contentActivity)
                // 데코레이션
                val deco = MaterialDividerItemDecoration(contentActivity, MaterialDividerItemDecoration.VERTICAL)
                addItemDecoration(deco)
            }
        }
    }
```

메서드를 호출해준다.

```
[ReadContentBottomFragment - onCreateView]

settingRecyclerViewAddContentReply()
```

액정의 높이를 구하는 메서드를 작성해준다.

```
[ReadContentBottomFragmeht.kt]

    // 사용자 단말기 액정의 길이를 구해 반환하는 메서드
    fun getWindowHeight() : Int {
        // 화면 크기 정보를 담을 배열 객체
        val displayMetrics = DisplayMetrics()
        // 액정의 가로 세로 길이 정보를 담아준다.
        contentActivity.windowManager.defaultDisplay.getMetrics(displayMetrics)
        // 세로길이를 반환해준다.
        return displayMetrics.heightPixels
    }
```

액정 높이의 85% 만큼을 구하는 메서드를 작성해준다.

```
[ReadContentBottomFragment.kt]

    // BottomSheet의 높이를 구한다(화면 액정의 85% 크기)
    fun getBottomSheetDialogHeight() : Int {
        return (getWindowHeight() * 0.85).toInt()
    }
```

BottomSheet의 높이를 설정하는 메서드를 호출해준다.

```
[ReadContentBottomFragmeht.kt]

    // BottomSheet의 높이를 설정해준다.
    fun setBottomSheetHeight(bottomSheetDialog:BottomSheetDialog){
        // BottomSheet의 기본 뷰 객체를 가져온다
        val bottomSheet = bottomSheetDialog.findViewById<View>(com.google.android.material.R.id.design_bottom_sheet)!!
        // BottomSheet 높이를 설정할 수 있는 객체를 생성한다.
        val behavior = BottomSheetBehavior.from(bottomSheet)
        // 높이를 설정한다.
        val layoutParams = bottomSheet.layoutParams
        layoutParams.height = getBottomSheetDialogHeight()
        bottomSheet.layoutParams = layoutParams
        behavior.state = BottomSheetBehavior.STATE_EXPANDED

    }
```

BottomSheetDialog가 생성될 때 호출되는 메서드를 구현해주고 높이를 설정할 수 있도록 작성해준다.

```
[ReadContentBottomFragment.kt]

    // 다이얼로그가 만들어질 때 자동으로 호출되는 메서드
    override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
        // 다이얼로그를 받는다.
        val dialog = super.onCreateDialog(savedInstanceState)
        // 다이얼로그가 보일때 동작하는 리스너
        dialog.setOnShowListener {

            val bottomSheetDialog = it as BottomSheetDialog
            // 높이를 설정한다.
            setBottomSheetHeight(bottomSheetDialog)
        }

        return dialog
    }
```

013 ModifyContentFragment 작업

ModifyContentFragment 를 생성한다.

Fragment 이름을 등록해준다.

```
[Tools.kt]

// ContentActivity에서 보여줄 프래그먼트들의 이름
enum class ContentFragmentName(var str:String){
    MAIN_FRAGMENT("MainFragment"),
    ADD_CONTENT_FRAGMENT("AddContentFragment"),
    READ_CONTENT_FRAGMENT("ReadContentFragment"),
    MODIFY_CONTENT_FRAGMENT("ModifyContentFragment"),
}
```

Fragment 객체 생성 코드를 넣어준다.

```
[ContentActivity.kt - replaceFragment]

            // 게시글 수정 화면
            ContentFragmentName.MODIFY_CONTENT_FRAGMENT -> {
                newFragment = ModifyContentFragment()
            }
```

수정하기 메뉴를 누르면 ModifyContentFragment를 보이게 한다.

```
[ReadContentFragment.kt - settingToolbarReadContent]

                        // 수정하기
                        R.id.menuItemReadContentModify -> {
                            // 수정 화면이 보이게 한다.
                            contentActivity.replaceFragment(ContentFragmentName.MODIFY_CONTENT_FRAGMENT, true, true, null)
                        }
```

ModifyContentFragment의 화면을 구성해준다.

```
[fragment_modify_content.xml]

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:transitionGroup="true"
    tools:context=".fragment.ModifyContentFragment" >

    <com.google.android.material.appbar.MaterialToolbar
        android:id="@+id/toolbarModifyContent"
        style="@style/Theme.AndroidProject4BoardApp.Toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:minHeight="?attr/actionBarSize"
        android:theme="?attr/actionBarTheme" />

    <ScrollView
        android:layout_width="match_parent"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:padding="10dp" >

        <com.google.android.material.textfield.TextInputLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="제목"
            app:endIconMode="clear_text"
            app:startIconDrawable="@drawable/subject_24px">

            <com.google.android.material.textfield.TextInputEditText
                android:id="@+id/textFieldModifyContentSubject"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:inputType="text"
                android:textAppearance="@style/TextAppearance.AppCompat.Large" />
        </com.google.android.material.textfield.TextInputLayout>

        <com.google.android.material.button.MaterialButtonToggleGroup
            android:id="@+id/toggleModifyContentType"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            app:selectionRequired="true"
            app:singleSelection="true" >

            <Button
                android:id="@+id/buttonModifyContentType1"
                style="@style/Widget.Material3.Button.OutlinedButton"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="자유"
                android:textAppearance="@style/TextAppearance.AppCompat.Medium" />

            <Button
                android:id="@+id/buttonModifyContentType2"
                style="@style/Widget.Material3.Button.OutlinedButton"
                android:layout_width="match_parent"
```

```
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="유머"
            android:textAppearance="@style/TextAppearance.AppCompat.Medium" />

        <Button
            android:id="@+id/buttonModifyContentType3"
            style="@style/Widget.Material3.Button.OutlinedButton"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="시사"
            android:textAppearance="@style/TextAppearance.AppCompat.Medium" />

        <Button
            android:id="@+id/buttonModifyContentType4"
            style="@style/Widget.Material3.Button.OutlinedButton"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="스포츠"
            android:textAppearance="@style/TextAppearance.AppCompat.Medium" />
    </com.google.android.material.button.MaterialButtonToggleGroup>

    <com.google.android.material.textfield.TextInputLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:hint="내용"
        app:endIconMode="clear_text"
        app:startIconDrawable="@drawable/description_24px">

        <com.google.android.material.textfield.TextInputEditText
            android:id="@+id/textFieldModifyContentText"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:inputType="text|textMultiLine"
            android:textAppearance="@style/TextAppearance.AppCompat.Large" />
    </com.google.android.material.textfield.TextInputLayout>

    <Button
        android:id="@+id/buttonModifyContentImageDelete"
        style="@style/Widget.Material3.Button.OutlinedButton"
        android:layout_width="match_parent"
```

```
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginTop="10dp"
                android:text="이미지 삭제"
                android:textAppearance="@style/TextAppearance.AppCompat.Large" />

            <ImageView
                android:id="@+id/imageViewModifyContent"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginTop="10dp"
                android:adjustViewBounds="true"
                app:srcCompat="@drawable/panorama_24px" />


        </LinearLayout>
    </ScrollView>
</LinearLayout>
```

툴바 설정 메서드를 구현해준다.

```
[ModifyContentFragment.kt]

    // 툴바 설정
    fun settingToolbarModifyContent(){
        fragmentModifyContentBinding.apply {
            toolbarModifyContent.apply {
                // 타이틀
                title = "글 수정"
                // Back
                setNavigationIcon(R.drawable.arrow_back_24px)
                setNavigationOnClickListener {
                    contentActivity.removeFragment(ContentFragmentName.MODIFY_CONTENT_FRAGMENT)
                }
            }
        }
    }
```

메서드를 호출해준다.

```
[ModifyContentFragment.kt - onCreateView]

settingToolbarModifyContent()
```

메뉴 파일을 생성해준다.

```
[res/menu/menu_add_content.xml]

<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:id="@+id/menuItemModifyContentCamera"
        android:icon="@drawable/photo_camera_24px"
        android:title="카메라"
        app:showAsAction="always" />
    <item
        android:id="@+id/menuItemModifyContentAlbum"
        android:icon="@drawable/photo_album_24px"
        android:title="앨범"
        app:showAsAction="always" />
    <item
        android:id="@+id/menuItemModifyContentReset"
        android:icon="@drawable/clear_all_24px"
        android:title="초기화"
        app:showAsAction="always" />
    <item
        android:id="@+id/menuItemModifyContentDone"
        android:icon="@drawable/done_24px"
        android:title="완료"
        app:showAsAction="always" />
</menu>
```

메뉴를 보여주는 코드를 넣어준다.

```
[ModifyContentFragment.kt - settingToolbarModifyContent]

                // 메뉴
                inflateMenu(R.menu.menu_modify_content)
```

014 ModifyUserFragment 작업

ModifyUserFragment 를 생성한다.

Fragment의 이름을 등록해준다.

```
[Tools.kt]

// ContentActivity에서 보여줄 프래그먼트들의 이름
enum class ContentFragmentName(var str:String){
    MAIN_FRAGMENT("MainFragment"),
    ADD_CONTENT_FRAGMENT("AddContentFragment"),
    READ_CONTENT_FRAGMENT("ReadContentFragment"),
    MODIFY_CONTENT_FRAGMENT("ModifyContentFragment"),
    MODIFY_USER_FRAGMENT("ModifyUserFragment"),
}
```

Fragment 생성 코드를 넣어준다.

```
[ContentActivity.kt - replaceFragment]

            // 사용자 정보 수정 화면
            ContentFragmentName.MODIFY_USER_FRAGMENT -> {
                newFragment = ModifyUserFragment()
            }
```

사이드 메뉴를 누르면 동작하는 부분을 구현해준다.

```
[ContentActivity.kt - settingNavigationView]

                    // 메뉴의 id로 분기한다.
```

```
                when(it.itemId){
                    // 전체 게시판
                    R.id.menuItemContentNavigationAll -> {
                        // NavigationView를 닫아준다.
                        drawerLayoutContent.close()
                    }
                    // 자유 게시판
                    R.id.menuItemContentNavigation1 -> {
                        // NavigationView를 닫아준다.
                        drawerLayoutContent.close()
                    }
                    // 유머 게시판
                    R.id.menuItemContentNavigation2 -> {
                        // NavigationView를 닫아준다.
                        drawerLayoutContent.close()
                    }
                    // 시사 게시판
                    R.id.menuItemContentNavigation3 -> {
                        // NavigationView를 닫아준다.
                        drawerLayoutContent.close()
                    }
                    // 스포츠 게시판
                    R.id.menuItemContentNavigation4 -> {
                        // NavigationView를 닫아준다.
                        drawerLayoutContent.close()
                    }
                    // 사용자 정보 수정
                    R.id.menuItemContentNavigationModifyUserInfo -> {
                        replaceFragment(ContentFragmentName.MODIFY_USER_FRAGMENT, false, false, null)
                        // NavigationView를 닫아준다.
                        drawerLayoutContent.close()
                    }
                    // 로그아웃
                    R.id.menuItemContentNavigationLogout -> {

                    }
                    // 회원 탈퇴
                    R.id.menuItemContentNavigationSignOut -> {

                    }
                }
```

화면을 구성해준다.

```
[res/layout/fragment_modify_user.xml]

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:transitionGroup="true"
    tools:context=".fragment.ModifyUserFragment">

    <com.google.android.material.appbar.MaterialToolbar
        android:id="@+id/toolbarModifyUser"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:minHeight="?attr/actionBarSize"
        android:theme="?attr/actionBarTheme" />

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical"
            android:padding="10dp">

            <com.google.android.material.textfield.TextInputLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:hint="닉네임"
                app:endIconMode="clear_text"
                app:startIconDrawable="@drawable/person_add_24px">

                <com.google.android.material.textfield.TextInputEditText
                    android:id="@+id/textFieldModifyUserInfoNickName"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:inputType="text"
```

```
        android:textAppearance="@style/TextAppearance.AppCompat.Large" />
</com.google.android.material.textfield.TextInputLayout>

<com.google.android.material.textfield.TextInputLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp"
    android:hint="나이"
    app:endIconMode="clear_text"
    app:startIconDrawable="@drawable/face_24px">

    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/textFieldModifyUserInfoAge"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="number|numberDecimal"
        android:textAppearance="@style/TextAppearance.AppCompat.Large" />
</com.google.android.material.textfield.TextInputLayout>

<com.google.android.material.textfield.TextInputLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp"
    android:hint="비밀번호"
    app:endIconMode="password_toggle"
    app:startIconDrawable="@drawable/key_24px">

    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/textFieldModifyUserPw"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="text|textPassword"
        android:textAppearance="@style/TextAppearance.AppCompat.Large" />
</com.google.android.material.textfield.TextInputLayout>

<com.google.android.material.textfield.TextInputLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp"
    android:hint="비밀번호 확인"
    app:endIconMode="password_toggle"
    app:startIconDrawable="@drawable/key_24px">

    <com.google.android.material.textfield.TextInputEditText
```

```
            android:id="@+id/textFieldModifyUserPw2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:inputType="text|textPassword"
            android:textAppearance="@style/TextAppearance.AppCompat.Large" />
    </com.google.android.material.textfield.TextInputLayout>

    <com.google.android.material.button.MaterialButtonToggleGroup
        android:id="@+id/toggleModifyUserInfoGender"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        app:selectionRequired="true"
        app:singleSelection="true">

        <Button
            android:id="@+id/buttonModifyUserInfoMale"
            style="@style/Widget.Material3.Button.OutlinedButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="남자"
            android:textAppearance="@style/TextAppearance.AppCompat.Large" />

        <Button
            android:id="@+id/buttonModifyUserInfoFemale"
            style="@style/Widget.Material3.Button.OutlinedButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="여자"
            android:textAppearance="@style/TextAppearance.AppCompat.Large" />

    </com.google.android.material.button.MaterialButtonToggleGroup>

    <com.google.android.material.checkbox.MaterialCheckBox
        android:id="@+id/checkBoxModifyUserInfoAll"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:text="취미"
        android:textAppearance="@style/TextAppearance.AppCompat.Large" />

    <LinearLayout
        android:id="@+id/checkBoxGroupModifyUserInfo1"
```

```
        android:id="@+id/checkBoxGroupModifyUserInfo1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginStart="10dp"
        android:layout_marginTop="10dp"
        android:orientation="horizontal">

        <com.google.android.material.checkbox.MaterialCheckBox
            android:id="@+id/checkBoxModifyUserInfoHobby1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="운동" />

        <com.google.android.material.checkbox.MaterialCheckBox
            android:id="@+id/checkBoxModifyUserInfoHobby2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="독서" />

        <com.google.android.material.checkbox.MaterialCheckBox
            android:id="@+id/checkBoxModifyUserInfoHobby3"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="영화감상" />

    </LinearLayout>

    <LinearLayout
        android:id="@+id/checkBoxGroupModifyUserInfo2"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginStart="10dp"
        android:layout_marginTop="10dp"
        android:orientation="horizontal">

        <com.google.android.material.checkbox.MaterialCheckBox
            android:id="@+id/checkBoxModifyUserInfoHobby4"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="요리" />
```

```
                <com.google.android.material.checkbox.MaterialCheckBox
                    android:id="@+id/checkBoxModifyUserInfoHobby5"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:layout_weight="1"
                    android:text="음악" />

                <com.google.android.material.checkbox.MaterialCheckBox
                    android:id="@+id/checkBoxModifyUserInfoHobby6"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:layout_weight="1"
                    android:text="기타" />

            </LinearLayout>

        </LinearLayout>

    </ScrollView>

</LinearLayout>
```

메뉴를 구성해준다.

```
[res/menu/menu_modify_user.xml]

<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:id="@+id/menuItemModifyUserDone"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:icon="@drawable/done_24px"
        android:title="완료"
        app:showAsAction="always" />
</menu>
```

ModifyUserFragment 에 기본 코드를 넣어준다.

```
[ModifyUserFragment.kt]

    lateinit var fragmentModifyContentBinding: FragmentModifyContentBinding
    lateinit var contentActivity: ContentActivity

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {
        // Inflate the layout for this fragment

        fragmentModifyContentBinding = FragmentModifyContentBinding.inflate(inflater)
        contentActivity = activity as ContentActivity

        return fragmentModifyContentBinding.root
    }
```

툴바를 설정하는 메서드를 구현한다.

```
    // 툴바 설정
    fun settingToolbarModifyUser(){
        fragmentModifyContentBinding.apply {
            toolbarModifyContent.apply {
                // 타이틀
                title = "회원 정보 수정"
                // 메뉴
                inflateMenu(R.menu.menu_modify_user)
            }
        }
    }
```

메서드를 호출한다.

```
[ModifyUserFragment.kt - onCreateView]

        settingToolbarModifyUser()
```

015 MVVM 적용

build.gradle.kts 에서 viewBinding 을 dataBinding으로 변경

```
[build.gradle.kts]

    buildFeatures{
        // viewBinding = true
        dataBinding = true
    }
```

/res/layout 폴더에 있는 모든 xml 파일을 태그로묶어준다.

```
<layout>
</layout>
```

016 회원가입화면 작업

viewmodel 패키지를 만들어준다

viewmodel 패키지에 JoinViewModel 을 만들어준다

UI 요소와 연결될 MutableLiveData 들을 정의해준다.

```
[JoinViewModel.kt]

package kr.co.lion.androidproject4boardapp.viewmodel

import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel

class JoinViewModel : ViewModel(){
    // 아이디
    val textFieldJoinUserId = MutableLiveData<String>()
    // 비밀번호
    val textFieldJoinUserPw = MutableLiveData<String>()
    // 비밀번호2
    val textFieldJoinUserPw2 = MutableLiveData<String>()
}
```

res/layout/fragment_join.xml 에 ViewModel 객체 정의

```
[res/layout/fragment_join.xml]

    <data>
        <variable
            name="joinViewModel"
            type="kr.co.lion.androidproject4boardapp.viewmodel.JoinViewModel" />
    </data>
```

res/layout/fragment_join.xml 에 ViewModel이 가지고 있는 MutableLiveData와와 동기화 시킨다.

@= : 양방향. MutableLiveData의 값을 변경했을 때 xml의 속성값이 변경되고 xml의 속성값이 변경되었을 때 MutableLiveData의 값도 변경된다

@ : 단방향. MutableLiveData의 값을 변경했을 때 xml의 속성값이 변경되는 것만 가능하다.

```
[res/layout/fragment_join.xml]

모든 TextInputEditText 에 넣어준다.

android:text="@={joinViewModel.textFieldJoinUserId}"

android:text="@={joinViewModel.textFieldJoinUserPw}"

android:text="@={joinViewModel.textFieldJoinUserPw2}"
```

ViewModel 객체를 생성하고 Binding 객체에 설정해준다.

```
[JoinFragment.kt]

    lateinit var joinViewModel: JoinViewModel

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {
        // Inflate the layout for this fragment

        // fragmentJoinBinding = FragmentJoinBinding.inflate(inflater)
        // 바인딩 객체를 생성한다. ViewBinding의 기능을 포함한다
        // 첫 번째 : LayoutInflater
        // 두 번째 : 화면을 만들 때 사용할 layout폴더의 xml 파일
        // 세 번째 : xml 을 통해서 만들어진 화면을 누가 관리하게 할 것인가. 여기서는 Fragment를 의미한다.
        // 네 번째 : Fragment 상태에 영향을 받을 것인가...
        fragmentJoinBinding = DataBindingUtil.inflate(inflater, R.layout.fragment_join, container, false)
        // ViewModel 객체를 생성한다.
        joinViewModel = JoinViewModel()
        // 생성한 ViewModel 객체를 layout 파일에 설정해준다.
        fragmentJoinBinding.joinViewModel = joinViewModel
        // ViewModel의 생명 주기를 Fragment와 일치시킨다. Fragment가 살아 있을 때 ViewModel 객체도 살아 있겠금 해준다.
        fragmentJoinBinding.lifecycleOwner = this

        mainActivity = activity as MainActivity

        settingToolbar()
        settingButtonJoinNext()

        return fragmentJoinBinding.root
    }
```

툴바의 타이틀과 연결될 MutableLiveData를 추가해준다.

```
[JoinViewModel.kt]

    // 툴바의 타이틀
    val toolbarJoinTitle = MutableLiveData<String>()
```

툴바의 title 속성과 MutableLiveData를 동기화 시켜준다.

```
[res/layout/fragment_join.xml]

    <com.google.android.material.appbar.MaterialToolbar
        (생략)
        app:title="@{joinViewModel.toolbarJoinTitle}"/>
```

Tools.kt 에 키보드 올리는 메서드, 키보드 내리는 메서드, 경고 다이얼로그를 띄우는 메서드를 추가한다.

```
[Tool.kt]

// 뷰에 포커스를 주고 키보드를 올린다.
        fun showSoftInput(context: Context, view: View){
            // 뷰에 포커스를 준다.
            view.requestFocus()
            thread {
                // 딜레이
                SystemClock.sleep(200)
                // 키보드 관리 객체를 가져온다.
                val inputMethodManger = context.getSystemService(AppCompatActivity.INPUT_METHOD_SERVICE) as
InputMethodManager
                // 키보드를 올린다.
                inputMethodManger.showSoftInput(view, 0)
            }
        }

        // 키보드를 내려주고 포커스를 제거한다.
        fun hideSoftInput(activity: Activity){
            // 포커스를 가지고 있는 뷰가 있다면..
            if(activity.window.currentFocus != null){
                // 키보드 관리 객체를 가져온다.
                val inputMethodManger = activity.getSystemService(AppCompatActivity.INPUT_METHOD_SERVICE) as
InputMethodManager
                // 키보드를 내려준다.
                inputMethodManger.hideSoftInputFromWindow(activity.window.currentFocus?.windowToken, 0)
                // 포커스를 제거해준다.
                activity.window.currentFocus?.clearFocus()
            }
        }

        // 입력 요소가 비어있을때 보여줄 다이얼로그를 구성하는 메서드
        fun showErrorDialog(context: Context, view: View, title:String, message:String){
            val materialAlertDialogBuilder = MaterialAlertDialogBuilder(context)
            materialAlertDialogBuilder.setTitle(title)
            materialAlertDialogBuilder.setMessage(message)
            materialAlertDialogBuilder.setPositiveButton("확인"){ dialogInterface: DialogInterface, i: Int ->
                showSoftInput(context, view)
            }
            materialAlertDialogBuilder.show()
        }
```

텍스트 필드 초기 설정 메서드를 만들어준다.

```
[JoinFragment.kt]

    // 입력요소 초기설정
    fun settingTextField(){
        // 입력 요소들을 초기화 한다.
        joinViewModel.textFieldJoinUserId.value = ""
        joinViewModel.textFieldJoinUserPw.value = ""
        joinViewModel.textFieldJoinUserPw2.value = ""
        // 첫 번째 입력 요소에 포커스를 준다.
        Tools.showSoftInput(mainActivity, fragmentJoinBinding.textFieldJoinUserId)
    }
```

메서드를 호출해준다.

```
[JoinFragment.kt - onCreateView]
        settingTextField()
```

입력 유효성 검사 메서드를 구현해준다.

[JoinFragment.kt]

```
    // 입력요소 유효성 검사 메서드
    fun checkTextInput():Boolean{

        // 사용자가 입력한 내용을 가져온다
        val userId = joinViewModel.textFieldJoinUserId.value!!
        val userPw = joinViewModel.textFieldJoinUserPw.value!!
        val userPw2 = joinViewModel.textFieldJoinUserPw2.value!!

        // 아이디를 입력하지 않았다면
        if(userId.isEmpty()){
            Tools.showErrorDialog(mainActivity, fragmentJoinBinding.textFieldJoinUserId, "아이디 입력 오류",
                "아이디를 입력해주세요")
            return false
        }

        // 비밀번호를 입력하지 않았다면
        if(userPw.isEmpty()){
            Tools.showErrorDialog(mainActivity, fragmentJoinBinding.textFieldJoinUserPw, "비밀번호 입력 오류",
                "비밀번호를 입력해주세요")
            return false
        }

        // 비밀번호 확인을 입력하지 않았다면
        if(userPw2.isEmpty()){
            Tools.showErrorDialog(mainActivity, fragmentJoinBinding.textFieldJoinUserPw2, "비밀번호 입력 오류",
                "비밀번호를 입력해주세요")
            return false
        }

        // 입력한 비밀번호가 서로 다르다면
        if(userPw != userPw2){
            joinViewModel.textFieldJoinUserPw.value = ""
            joinViewModel.textFieldJoinUserPw2.value = ""
            Tools.showErrorDialog(mainActivity, fragmentJoinBinding.textFieldJoinUserPw, "비밀번호 입력 오류",
                "비밀번호가 다릅니다")
            return false
        }

        return true
    }
```

다음 버튼을 눌렀을 때 메서드를 호출해준다.

```
[JoinFragment - settingButtonJoinNext]

                // 버튼을 눌렀을 때
                setOnClickListener  {
                    // 입력을 검사한다.
                    val chk = checkTextInput()

                    // 입력이 모두 잘 되어 있다면..
                    if(chk == true) {
                        // 키보를 내려준다.
                        Tools.hideSoftInput(mainActivity)
                        // AddUserInfoFragment를 보여준다.
                        mainActivity.replaceFragment(MainFragmentName.ADD_USER_INFO_FRAGMENT, true, true, null)
                    }
                }
```

아이디 중복 확인 여부를 검사하기 위한 변수를 정의해준다.

```
[JoinFragment]

    // 아이디 중복 확인 검사를 했는지..
    // true면 아이디 중복 확인 검사를 완료한 것으로 취급한다.
    var checkUserIdExist = false
```

아이디를 입력하면 변수에 **false**를 넣어준다.

```
[JoinFragment - settingTextField]

        // 아이디 입력요소의 값을 변경하면 중복확인 여부 변수값을 false로 설정한다.
        fragmentJoinBinding.textFieldJoinUserId.addTextChangedListener {
            checkUserIdExist = false
        }
```

입력 유효성 검사에 아이디 중복 확인 여부를 검사하는 코드를 넣어준다.

```
[JoinFragment - checkTextInput]

        // 아이디 중복확인을 하지 않았다면..
        if(checkUserIdExist == false){
            Tools.showErrorDialog(mainActivity, fragmentJoinBinding.textFieldJoinUserId, "아이디 중복 확인 오류",
                "아이디 중복확인을 해주세요")
            return false
        }
```

중복 확인 버튼 이벤트 처리 코드를 넣어준다.

```
    // 중복확인 버튼
    fun settingButtonJoinCheckId(){
        fragmentJoinBinding.apply {
            buttonJoinCheckId.apply {
                setOnClickListener {
                    checkUserIdExist = true
                }
            }
        }
    }
```

메서드를 호출한다.

```
[JoinFragment - onCreateView]

settingButtonJoinCheckId()
```

AddUserInfoViewModel 클래스를 생성한다.

남자, 여자를 나타내는 값을 정의한다

```
[Tools.kt]

// 남자 또는 여자를 나타내는 값을 정의한다.
enum class Gender(var str:String){
    MALE("male"),
    FEMALE("female")
}
```

AddUserInfoViewModel 클래스를 작성해준다.

```
package kr.co.lion.androidproject4boardapp.viewmodel

import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import kr.co.lion.androidproject4boardapp.Gender
import kr.co.lion.androidproject4boardapp.R

class AddUserInfoViewModel : ViewModel(){

    // 닉네임
    val textFieldAddUserInfoNickName = MutableLiveData<String>()
    // 나이
    val textFieldAddUserInfoAge = MutableLiveData<String>()
    // 성별
    val toggleAddUserInfoGender = MutableLiveData<Int>()

    // 성별을 셋팅하는 메서드
    fun settingGender(gender:Gender){
        // 성별로 분기한다.
        when(gender){
            Gender.MALE -> {
                toggleAddUserInfoGender.value = R.id.buttonAddUserInfoMale
            }
            Gender.FEMALE -> {
                toggleAddUserInfoGender.value = R.id.buttonAddUserInfoFemale
            }
        }
    }

}
```

layout xml 에 ViewModel 객체 정의

```
[res/fragment_add_user_info.xml]

    <data>
        <variable
            name="addUserInfoViewModel"
            type="kr.co.lion.androidproject4boardapp.viewmodel.AddUserInfoViewModel" />
    </data>
```

각 입력요소에 ViewModel의 LiveData를 연결시켜준다.

```
[res/fragment_add_user_info.xml]

android:text="@={addUserInfoViewModel.textFieldAddUserInfoNickName}"
android:text="@={addUserInfoViewModel.textFieldAddUserInfoAge}"
app:checkedButton="@={addUserInfoViewModel.toggleAddUserInfoGender}"
```

Fragment에 ViewModel 코드를 작성해준다.

```
[AddUserInfoFragment.kt]


    lateinit var addUserInfoViewModel: AddUserInfoViewModel

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {
        // Inflate the layout for this fragment

        // fragmentAddUserInfoBinding = FragmentAddUserInfoBinding.inflate(inflater)
        fragmentAddUserInfoBinding = DataBindingUtil.inflate(inflater, R.layout.fragment_add_user_info, container, false)
        addUserInfoViewModel = AddUserInfoViewModel()
        fragmentAddUserInfoBinding.addUserInfoViewModel = addUserInfoViewModel
        fragmentAddUserInfoBinding.lifecycleOwner = this

        mainActivity = activity as MainActivity

        settingToolbar()
        settingButtonAddUserInfoSubmit()

        return fragmentAddUserInfoBinding.root
    }
```

입력 요소 설정 메서드를 구현해준다.

```
[AdduserFragment.kt]

    // 입력 요소 관련 설정
    fun settingInputUI(){
        // 입력 요소들을 초기화 한다.
        addUserInfoViewModel.textFieldAddUserInfoNickName.value = ""
        addUserInfoViewModel.textFieldAddUserInfoAge.value = ""
        addUserInfoViewModel.settingGender(Gender.MALE)
        // 닉네임에 포커스를 준다.
        Tools.showSoftInput(mainActivity, fragmentAddUserInfoBinding.textFieldAddUserInfoNickName)
    }
```

메서드를 호출해준다.

```
[AddUserFragment.kt - onCreateview]

        settingInputUI()
```

실행해보면 다음과 같은 오류가 발생한다.

```
import kr.co.lion.androidproject4boardapp.databinding.FragmentAddUserInfoBindingImpl;
                                                      ^
  symbol:   class FragmentAddUserInfoBindingImpl
  location: package kr.co.lion.androidproject4boardapp.databinding
```

위의 메시지 처럼 머머머머.databinding.머머머... 이런 메시지는 layout xml 파일이 잘못되어있을 경우 발생하는 오류이다.

안드로이드 스튜디오에 나오는 오류 메시지 중 제일 아래 것을 눌러서 어떤 UI 요소에 대한 오류인지 확인한다.

"msg":"Cannot find a getter 라는 메시지가 나오면 MutableLiveData 를 셋팅한 속성이 단방향만 지원되는 경우이다. 이때는 @= 를 @ 로 변경해준다.

```
[fragment_add_user_info.xml]

app:checkedButton="@{addUserInfoViewModel.toggleAddUserInfoGender}"
```

수정 후 실행해보면 또 오류가 난다.

"msg":"Cannot find a setter 라는 메시지가 나오면 MutableLiveData 를 셋팅할 수 없는 속성이다.

이때는 다른 속성을 찾아서 변경해주던가 속성을 만들어써야 한다.

속성을 찾는 방법은 kt 파일에서 setter/getter 메서드를 찾아준다.

AddUserInfoViewModel 에 새로운 속성에 대해 값을 넣어주는 작업을 처리할 메서드를 만들어준다.

```
[AddUserInfoViewModel.kt]

    companion object {
        // ViewModel에 값을 설정하여 화면에 반영하는 작업을 할 때 호출된다.
        // () 안에는 속성의 이름을 넣어준다. 속성의 이름은 자유롭게 해주면 되지만 기존의 속성 이름과 중복되지 않게
        // 해줘야 한다.
        // 매개변수 : 값이 설정된 View 객체, ViewModel을 통해 설정되는 값
        @BindingAdapter("android:checkedButtonId")
        @JvmStatic
        fun setCheckedButtonId(group:MaterialButtonToggleGroup, buttonId:Int){
            group.check(buttonId)
        }
    }
```

build.grade.kts 에 plugins에 다음과 같이 설정해준다.

```
    id("kotlin-kapt")
```

버튼 토글 그룹에 만든 속성을 설정해준다.

```
[fragment_add_user_info.xml]

            <com.google.android.material.button.MaterialButtonToggleGroup
            (생략)
                android:checkedButtonId="@{addUserInfoViewModel.toggleAddUserInfoGender}">
```

없는 속성을 만들어 MutableLiveData에 값을 셋팅할 때 동작(순방향)...

1. toggleAddUserInfoGender 라는 MutableLiveData에 버튼의 ID를 넣어줬다.

```
        // 성별로 분기한다.
        when(gender){
            Gender.MALE -> {
                toggleAddUserInfoGender.value = R.id.buttonAddUserInfoMale
            }
            Gender.FEMALE -> {
                toggleAddUserInfoGender.value = R.id.buttonAddUserInfoFemale
            }
        }
```

2. toggleAddUserInfoGender 라는 MutableLiveData 는 checkedButtonId 라는 속성과 연결되어 있다.

```
android:checkedButtonId="@{addUserInfoViewModel.toggleAddUserInfoGender}"
```

3. toggleAdduserInfoGender 라는 MutableLiveData에 값을 설정하면 이 값은 checkedButtonId 라는 속성에 설정된다.
4. 근데, 이 속성은 원래 없는 속성이고 우리가 ViewModel을 통해 만들어준 속성이다.

```
@BindingAdapter("android:checkedButtonId")
```

5. 속성에 값을 넣어주면 ViewModel 에 만들어놓은 메서드가 호출되고 첫 번째 매개변수에는 속성이 설정되어 있는 View 객체, 두 번째는 속성에 넣은 값이 들어온다. 이 두 매개변수를 이용해 필요한 작업을 한다.

```
        @BindingAdapter("android:checkedButtonId")
        @JvmStatic
        fun setCheckedButtonId(group:MaterialButtonToggleGroup, buttonId:Int){
            group.check(buttonId)
        }
```

역방향 설정관련

1. 속성값이 정의 되어 있는 InverseBindingAdapter 를 찾는다.

```
@InverseBindingAdapter(attribute = "android:checkedButtonId", event="checkedButtonChangeListener")
```

2. 찾은 InverseBindingAdapter 에 event에 설정되어 있는 이름을 확인한다.

```
event="checkedButtonChangeListener"
```

3. 확인한 이름으로 등록되어 있는 @BindingAdapter를 찾아 메서드를 호출한다.

```
@BindingAdapter("checkedButtonChangeListener")
@JvmStatic
fun checkedButtonChangeListener(group:MaterialButtonToggleGroup, inverseBindingListener:InverseBindingListener){
    group.addOnButtonCheckedListener { group, checkedId, isChecked ->
        inverseBindingListener.onChange()
    }
}
```

4. 호출되는 메서드의 첫번째 매개변수에는 속성이 설정되어 있는 View 객체의 주소값이 들어온다. 이를 이용해 View의 이벤트 리스너를 설정해주고 사건이 벌어졌을 때 동작하는 코드는 두 번재 매개변수로 들어오는 InverseBindingListener 객체의 onChange 메서드를 호출하는 코드를 작성해준다.

역방향이 발생하면(사용자 조작에 의해 속성값이 변경될때)

1. 위에서 설정된 View의 리스너가 동작한다.

```
    group.addOnButtonCheckedListener { group, checkedId, isChecked ->
        inverseBindingListener.onChange()
    }
```

2. 리스너에는 inverseBindingListener.onChange() 메서드를 호출하는 것으로 되어있다.

3. onChage 메서드가 호출되면 해당 속성이 등록되어 있는 InverseBindingAdapter 에 등록되어 잇는 메서드를 호출한다.

```
@InverseBindingAdapter(attribute = "android:checkedButtonId", event="checkedButtonChangeListener")
@JvmStatic
fun getCheckedButtonId(group:MaterialButtonToggleGroup):Int{
    return group.checkedButtonId
}
```

4. 이 메서드 반화하는 값이 MutableLiveData에 설정된다.

취미들의 속성과 연결할 MutableLiveData를 정의해준다.

```
[AddUIserInfoViewModel.kt]

    // 취미들
    val checkBoxAddUserInfoHobby1 = MutableLiveData<Boolean>()
    val checkBoxAddUserInfoHobby2 = MutableLiveData<Boolean>()
    val checkBoxAddUserInfoHobby3 = MutableLiveData<Boolean>()
    val checkBoxAddUserInfoHobby4 = MutableLiveData<Boolean>()
    val checkBoxAddUserInfoHobby5 = MutableLiveData<Boolean>()
    val checkBoxAddUserInfoHobby6 = MutableLiveData<Boolean>()
```

checkbox의 checked 속성과 MutableLiveData를 연결해준다.

```
            <LinearLayout
                android:id="@+id/checkBoxGroupAddUserInfo1"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:layout_marginStart="10dp"
                android:layout_marginTop="10dp"
                android:orientation="horizontal">

                <com.google.android.material.checkbox.MaterialCheckBox
                    android:id="@+id/checkBoxAddUserInfoHobby1"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:layout_weight="1"
                    android:text="운동"
                    android:checked="@={addUserInfoViewModel.checkBoxAddUserInfoHobby1}"/>

                <com.google.android.material.checkbox.MaterialCheckBox
                    android:id="@+id/checkBoxAddUserInfoHobby2"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:layout_weight="1"
                    android:text="독서"
                    android:checked="@={addUserInfoViewModel.checkBoxAddUserInfoHobby2}"/>

                <com.google.android.material.checkbox.MaterialCheckBox
                    android:id="@+id/checkBoxAddUserInfoHobby3"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:layout_weight="1"
                    android:text="영화감상"
```

```
            android:text="영화감상"
            android:checked="@={addUserInfoViewModel.checkBoxAddUserInfoHobby3}"/>

    </LinearLayout>

    <LinearLayout
        android:id="@+id/checkBoxGroupAddUserInfo2"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginStart="10dp"
        android:layout_marginTop="10dp"
        android:orientation="horizontal">

        <com.google.android.material.checkbox.MaterialCheckBox
            android:id="@+id/checkBoxAddUserInfoHobby4"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="요리"
            android:checked="@={addUserInfoViewModel.checkBoxAddUserInfoHobby4}"/>

        <com.google.android.material.checkbox.MaterialCheckBox
            android:id="@+id/checkBoxAddUserInfoHobby5"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="음악"
            android:checked="@={addUserInfoViewModel.checkBoxAddUserInfoHobby5}"/>

        <com.google.android.material.checkbox.MaterialCheckBox
            android:id="@+id/checkBoxAddUserInfoHobby6"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="기타"
            android:checked="@={addUserInfoViewModel.checkBoxAddUserInfoHobby6}"/>

    </LinearLayout>
```

체크박스 초기 설정 메서드를 만들어준다.

```
[AddUserInfoFragment.kt]


    // 체크박스 관련 설정
    fun settingCheckBox(){
        // 모든 체크박스를 초기화한다.
        addUserInfoViewModel.checkBoxAddUserInfoHobby1.value = false
        addUserInfoViewModel.checkBoxAddUserInfoHobby2.value = false
        addUserInfoViewModel.checkBoxAddUserInfoHobby3.value = false
        addUserInfoViewModel.checkBoxAddUserInfoHobby4.value = false
        addUserInfoViewModel.checkBoxAddUserInfoHobby5.value = false
        addUserInfoViewModel.checkBoxAddUserInfoHobby6.value = false
    }
```

메서드를 호출한다.

```
[AddUserInfoFragment.kt - onCreaetView]

        settingCheckBox()
```

취미 전체와 연결할 MutableLiveData를 만들어준다

```
[AddUserInfoViewModel.kt]

    // 취미 전체
    val checkBoxAddUserInfoAllState = MutableLiveData<Int>()
    val checkBoxAddUserInfoAll = MutableLiveData<Boolean>()
```

화면 xml 파일에 속성과 MutableLiveData와 연결해준다.

```
[res/layout/fragment_add_user_info.xml]

            <com.google.android.material.checkbox.MaterialCheckBox
                (생략)
                app:checkedState="@{addUserInfoViewModel.checkBoxAddUserInfoAllState}"
                android:checked="@={addUserInfoViewModel.checkBoxAddUserInfoAll}"
                android:onClickListener="@{ (view) -> addUserInfoViewModel.onCheckBoxAllChanged()"/>
```

체크박스의 체크 상태 전체를 변경하는 메서드를 만들어준다.

```
[AddUserInfoViewModel.kt]

    // 체크박스 전체 상태를 설정하는 메서드
    fun setCheckAll(checked:Boolean){
        checkBoxAddUserInfoHobby1.value = checked
        checkBoxAddUserInfoHobby2.value = checked
        checkBoxAddUserInfoHobby3.value = checked
        checkBoxAddUserInfoHobby4.value = checked
        checkBoxAddUserInfoHobby5.value = checked
        checkBoxAddUserInfoHobby6.value = checked
    }
```

전체 취미 체크박스의 체크 상태가 변경되면 모든 체크박스의 상태를 변경해주는 메서드를 만들어준다. 이 메서드는 layout 파일의 취미 체크박스의 onClickListener에 설정되어 있다.

```
[AddUserInfoViewModel.kt]

    // 전체 취미 체크박스를 누르면...
    fun onCheckBoxAllChanged(){
        // 전체 취미 체크박스의 체크 여부를 모든 체크박스에 설정해준다.
        setCheckAll(checkBoxAddUserInfoAll.value!!)
    }
```

각 체크박스들에 대한 체크 기능을 구현해준다. 이 메서드는 layout 파일의 각 체크박스의 onClickListener에 설정되어 있다.

```
[AddUserInfoViewModel.kt]

    // 각 체크박스를 누르면...
    fun onCheckBoxChanged(){
        // 체크되어 있는 체크박스 개수를 담을 변수
        var checkedCnt = 0

        // 체크되어 있다면 체크되어 있는 체크박스의 개수를 1 증가시킨다.
        if(checkBoxAddUserInfoHobby1.value == true){
            checkedCnt++
        }
        if(checkBoxAddUserInfoHobby2.value == true){
            checkedCnt++
        }
```

```
        if(checkBoxAddUserInfoHobby3.value == true){
            checkedCnt++
        }
        if(checkBoxAddUserInfoHobby4.value == true){
            checkedCnt++
        }
        if(checkBoxAddUserInfoHobby5.value == true){
            checkedCnt++
        }
        if(checkBoxAddUserInfoHobby6.value == true){
            checkedCnt++
        }

        // 체크되어 있는 것이 없다면
        if(checkedCnt == 0){
            checkBoxAddUserInfoAll.value = false
            checkBoxAddUserInfoAllState.value = MaterialCheckBox.STATE_UNCHECKED
        }
        // 모두 체크되어 있다면
        else if(checkedCnt == 6){
            checkBoxAddUserInfoAll.value = true
            checkBoxAddUserInfoAllState.value = MaterialCheckBox.STATE_CHECKED
        }
        // 일부만 체크되어 있다면
        else {
            checkBoxAddUserInfoAllState.value = MaterialCheckBox.STATE_INDETERMINATE
        }

    }
```

각 입력 요소에 대해 유효성 검사를 하는 메서드를 만들어준다.

[AddUserInfoFragment.kt]

```
    // 입력 요소에 대한 유효성 검사
    fun checkInputForm():Boolean{
        // 입력한 값을 가져온다.
        val userNickName = addUserInfoViewModel.textFieldAddUserInfoNickName.value!!
        val userAge = addUserInfoViewModel.textFieldAddUserInfoAge.value!!

        // 입력하지 않은 것이 있을 경우 경고문을 띄운다.
        if(userNickName.isEmpty()){
            Tools.showErrorDialog(mainActivity, fragmentAddUserInfoBinding.textFieldAddUserInfoNickName,
                "닉네임 입력 오류", "닉네임을 입력해주세요")
            return false
        }

        if(userAge.isEmpty()){
            Tools.showErrorDialog(mainActivity, fragmentAddUserInfoBinding.textFieldAddUserInfoAge,
                "나이 입력 오류", "나이를 입력해주세요")
            return false
        }

        return true
    }
```

가입 완료 버튼을 눌렀을 때의 코드를 변경한다.

```
[AddUserInfoFragment.kt]

    // 가입 버튼
    fun settingButtonAddUserInfoSubmit(){
        fragmentAddUserInfoBinding.apply {
            buttonAddUserInfoSubmit.apply {
                // 눌렀을 때
                setOnClickListener {
                    // 유효성 검사를 수행한다.
                    val chk = checkInputForm()

                    // 모든 유효성 검사에 통과를 했다면..
                    if(chk == true) {
                        val materialAlertDialogBuilder = MaterialAlertDialogBuilder(mainActivity)
                        materialAlertDialogBuilder.setTitle("가입완료")
                        materialAlertDialogBuilder.setMessage("가입이 완료되었습니다\n로그인해주세요")
                        materialAlertDialogBuilder.setPositiveButton("확인") { dialogInterface: DialogInterface, i: Int ->
                            mainActivity.removeFragment(MainFragmentName.ADD_USER_INFO_FRAGMENT)
                            mainActivity.removeFragment(MainFragmentName.JOIN_FRAGMENT)
                        }
                        materialAlertDialogBuilder.show()
                    }
                }
            }
        }
    }
```

017 로그인 화면 작업

LoginViewModel 클래스를 생성한다.

MutableLiveData 들을 정의한다.

```
[LoginViewModel.kt]

class LoginViewModel : ViewModel() {
    // 아이디
    val textFieldLoginUserId = MutableLiveData<String>()
    // 비밀번호
    val textFieldLoginUserPw = MutableLiveData<String>()
    // 자동로그인
    val checkBoxLoginAuto = MutableLiveData<Boolean>()
}
```

Layout 파일에 ViewModel을 정의한다.

```
[fragment_login.xml]

    <data>
        <variable
            name="loginViewModel"
            type="kr.co.lion.androidproject4boardapp.viewmodel.LoginViewModel" />
    </data>
```

UI 요소와 MutableLiveData를 연결해준다.

```
[fragment_login.xml]

android:text="@={loginViewModel.textFieldLoginUserId}"
android:text="@={loginViewModel.textFieldLoginUserPw}"
android:checked="@={loginViewModel.checkBoxLoginAuto}"
```

ViewModel 관련 코드 작성

```
[LoginFragment.kt]

    lateinit var loginViewModel: LoginViewModel

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {
        // Inflate the layout for this fragment

        // fragmentLoginBinding = FragmentLoginBinding.inflate(inflater)
        fragmentLoginBinding = DataBindingUtil.inflate(layoutInflater, R.layout.fragment_login, container, false)
        loginViewModel = LoginViewModel()
        fragmentLoginBinding.loginViewModel = loginViewModel
        fragmentLoginBinding.lifecycleOwner = this
```

유효성 검사 메서드를 구현한다.

```
[LoginFragment.kt]

    // 유효성 검사
    fun checkInputForm():Boolean{
        // 입력한 값들을 가져온다.
        val userId = loginViewModel.textFieldLoginUserId.value!!
        val userPw = loginViewModel.textFieldLoginUserPw.value!!

        if(userId.isEmpty()){
            Tools.showErrorDialog(mainActivity, fragmentLoginBinding.textFieldLoginUserId, "아이디 입력 오류",
                "아이디를 입력해주세요")
            return false
        }

        if(userPw.isEmpty()){
            Tools.showErrorDialog(mainActivity, fragmentLoginBinding.textFieldLoginUserPw, "비밀번호 입력 오류",
                "비밀번호를 입력해주세요")
            return false
        }

        return true
    }
```

로그인 버튼을 눌렀을 때 코드를 변경한다.

```
[LoginFragment.kt - settingButtonLoginSubmit]

                    // 유효성 검사
                    val chk = checkInputForm()

                    // 모든 유효성 검사에 통과를 했다면
                    if(chk == true) {
                        // ContentActivity를 실행한다.
                        val contentIntent = Intent(mainActivity, ContentActivity::class.java)
                        startActivity(contentIntent)
                        // MainActivity를 종료한다.
                        mainActivity.finish()
                    }
```

MutableLiveData 초기화 메서드를 만들어준다.

```
[LoginFragment.kt ]

    // 입력 요소들 초기화
    fun settingInputForm(){
        loginViewModel.textFieldLoginUserId.value = ""
        loginViewModel.textFieldLoginUserPw.value = ""
    }
```

메서드를 호출한다.

```
[LoginFragment.kt - onCreateView]

        settingInputForm()
```

018 글작성 화면 작업

AddContentViewModel 클래스를 생성한다.

MutableLiveData를 정의한다.

```
[AddContentViewModel.kt]

    // 제목
    val textFieldAddContentSubject = MutableLiveData<String>()
    // 내용
    val textFieldAddContentText = MutableLiveData<String>()
```

layout 파일에 ViewModel 설정을 해준다.

```
[fragment_add_content.xml]

    <data>
        <variable
            name="addContentViewModel"
            type="kr.co.lion.androidproject4boardapp.viewmodel.AddContentViewModel" />
    </data>


    android:text="@={addContentViewModel.textFieldAddContentSubject}"

    android:text="@={addContentViewModel.textFieldAddContentText}"
```

각 게시판 타입을 나타나내는 이름을 정의한다.

```
[Tools.kt]

// 게시판 종류를 나타내는 값을 정의한다.
enum class ContentType(var str:String, var number:Int){
    TYPE_ALL("전체게시판", 0),
    TYPE_FREE("자유게시판", 1),
    TYPE_HUMOR("유머게시판", 2),
    TYPE_SOCIETY("시사게시판", 3),
    TYPE_SPORTS("스포츠게시판", 4)
}
```

게시판 종류에 따라 MutableLiveData에 값을 넣는 메서드를 만들어준다.

```
[AddContentViewModel.kt]

    // 게시판 종류를 받아 MutableLiveData에 설정하는 메서드
    fun settingContentType(contentType:ContentType){
        toggleAddContentType.value = when(contentType){
            ContentType.TYPE_ALL -> 0
            ContentType.TYPE_FREE -> R.id.buttonAddContentType1
            ContentType.TYPE_HUMOR -> R.id.buttonAddContentType2
            ContentType.TYPE_SOCIETY -> R.id.buttonAddContentType3
            ContentType.TYPE_SPORTS -> R.id.buttonAddContentType4
        }
    }
```

layout 파일의 MaterialButtonToggleGroup에 새로운 속성을 넣어주고 MutableLiveData와 연결한다.

```
[fragment_add_content.xml]

android:checkedButtonId="@={addContentViewModel.toggleAddContentType}"
```

MVVM에 관련된 메서드를 구현해준다

[AddContentViewModel.kt]

```
    companion object {
        // toggleAddContentType 에 값을 설정했을 때 checkedButtonId 속성에 값을 반영해 줄 때 호출(순방향)
        @BindingAdapter("android:checkedButtonId")
        @JvmStatic
        fun setCheckedButtonId(group: MaterialButtonToggleGroup, buttonId:Int){
            group.check(buttonId)
        }

        // 안드로이드 OS가 현재 화면을 구성하고자 할 때 InverseBindingAdapter에 등록되어 있는 event 값을 보고
        // event에 등록되어 있는 이름과 동일한 BindingAdapter를 찾아 메서드를 호출해준다.
        // 리스너 설정을 해준다.
        @BindingAdapter("checkedButtonChangeListener")
        @JvmStatic
        fun checkedButtonChangeListener(group:MaterialButtonToggleGroup, inverseBindingListener: InverseBindingListener){
            group.addOnButtonCheckedListener { group, checkedId, isChecked ->
                inverseBindingListener.onChange()
            }
        }

        // 안드로이드 OS가 현재 화면을 구성하고자 할 때 속성에 설정된 MutableLiveData가 양방향(@=)으로 되어 있을 경우
        // 참고할 데이터가 셋팅되어 있는 메서드
        // inverseBindingListener.onChange() 호출시 동작할 메서드
        @InverseBindingAdapter(attribute = "android:checkedButtonId", event="checkedButtonChangeListener")
        @JvmStatic
        fun getCheckedButtonId(group:MaterialButtonToggleGroup):Int{
            return group.checkedButtonId
        }
    }
```

Viewbinding관련 코드를 넣어준다.

```
[AddContentFragment.kt]

    lateinit var addContentViewModel: AddContentViewModel

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {
        // Inflate the layout for this fragment

        // fragmentAddContentBinding = FragmentAddContentBinding.inflate(inflater)
        fragmentAddContentBinding = DataBindingUtil.inflate(inflater, R.layout.fragment_add_content, container, false)
        addContentViewModel = AddContentViewModel()
        fragmentAddContentBinding.addContentViewModel = addContentViewModel
        fragmentAddContentBinding.lifecycleOwner = this
```

입력 요소 설정 메서드를 구현해준다.

```
[AddContentFragment.kt]

    // 입력 요소 설정
    fun settingInputForm(){
        addContentViewModel.textFieldAddContentSubject.value = ""
        addContentViewModel.textFieldAddContentText.value = ""
        addContentViewModel.settingContentType(ContentType.TYPE_FREE)

        Tools.showSoftInput(contentActivity, fragmentAddContentBinding.textFieldAddContentSubject)
    }
```

메서드를 호출해준다.

```
[AddContentFragment.kt - onCreateview]

settingInputForm()
```

019 카메라 관련 작업

res/xml 폴더에 file_path.xml 파일을 만들어준다.

외부 저장소 까지의 경로 데이터를 넣어준다.

```
[res/xml/file_path.xml]

    <external-path
        name="storage/emulated/0"
        path="."/>
```

AndroidManifest.xml 에 file_path 를 등록해준다.

```
[AndroidManifest.xml]

        <!-- 촬영한 사진을 저장하는 프로바이더 -->
        <provider
            android:authorities="kr.co.lion.androidproject4boardapp.file_provider"
            android:name="androidx.core.content.FileProvider"
            android:exported="false"
            android:grantUriPermissions="true">
            <meta-data
                android:name="android.support.FILE_PROVIDER_PATHS"
                android:resource="@xml/file_path"/>
        </provider>
```

촬영된 사진이 저장될 경로를 구하는 메서드를 구현한다(이미 있으면 생략)

```
[Tools.kt]

        // 촬영된 사진이 저장될 경로를 구해서 반환하는 메서드
        // authorities : AndroidManifest.xml에 등록한 File Provider의 이름
        fun getPictureUri(context:Context, authorities:String):Uri{
            // 촬영한 사진이 저장될 경로
            // 외부 저장소 중에 애플리케이션 영역 경로를 가져온다.
            val rootPath = context.getExternalFilesDir(null).toString()
            // 이미지 파일명을 포함한 경로
            val picPath = "${rootPath}/tempImage.jpg"
            // File 객체 생성
            val file = File(picPath)
            // 사진이 저장된 위치를 관리할 Uri 생성
            val contentUri = FileProvider.getUriForFile(context, authorities, file)

            return contentUri
        }
```

회전 각도값 구하는 메서드, 회전 시키는 메서드, 이미지 크기 조정하는 메서드를 구현해준다(있으면 생략)

```
[Tools.kt]

        ///// 카메라, 앨범 공통 ////////
        // 사진의 회전 각도값을 반환하는 메서드
        // ExifInterface : 사진, 영상, 소리 등의 파일에 기록한 정보
        // 위치, 날짜, 조리개값, 노출 정도 등등 다양한 정보가 기록된다.
        // ExifInterface 정보에서 사진 회전 각도값을 가져와서 그만큼 다시 돌려준다.
        fun getDegree(context:Context, uri:Uri) : Int {
            // 사진 정보를 가지고 있는 객체 가져온다.
            var exifInterface: ExifInterface? = null


            if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q){
                // 이미지 데이터를 가져올 수 있는 Content Provide의 Uri를 추출한다.
                // val photoUri = MediaStore.setRequireOriginal(uri)
                // ExifInterface 정보를 읽어올 스트림을 추출한다.

                val inputStream = context.contentResolver.openInputStream(uri)!!
                // ExifInterface 객체를 생성한다.
                exifInterface = ExifInterface(inputStream)
            } else {
                // ExifInterface 객체를 생성한다.
                exifInterface = ExifInterface(uri.path!!)
            }

            if(exifInterface != null){
                // 반환할 각도값을 담을 변수
                var degree = 0
                // ExifInterface 객체에서 회전 각도값을 가져온다.
                val ori = exifInterface.getAttributeInt(ExifInterface.TAG_ORIENTATION, -1)

                degree = when(ori){
                    ExifInterface.ORIENTATION_ROTATE_90 -> 90
                    ExifInterface.ORIENTATION_ROTATE_180 -> 180
                    ExifInterface.ORIENTATION_ROTATE_270 -> 270
                    else -> 0
                }

                return degree
            }
```

```
            return 0
        }

        // 회전시키는 메서드
        fun rotateBitmap(bitmap: Bitmap, degree:Float): Bitmap {
            // 회전 이미지를 생성하기 위한 변환 행렬
            val matrix = Matrix()
            matrix.postRotate(degree)

            // 회전 행렬을 적용하여 회전된 이미지를 생성한다.
            // 첫 번째 : 원본 이미지
            // 두 번째와 세번째 : 원본 이미지에서 사용할 부분의 좌측 상단 x, y 좌표
            // 네번째와 다섯번째 : 원본 이미지에서 사용할 부분의 가로 세로 길이
            // 여기에서는 이미지데이터 전체를 사용할 것이기 때문에 전체 영역으로 잡아준다.
            // 여섯번째 : 변환행렬. 적용해준 변환행렬이 무엇이냐에 따라 이미지 변형 방식이 달라진다.
            val rotateBitmap = Bitmap.createBitmap(bitmap, 0, 0, bitmap.width, bitmap.height, matrix, false)

            return rotateBitmap
        }

        // 이미지 사이즈를 조정하는 메서드
        fun resizeBitmap(bitmap: Bitmap, targetWidth:Int): Bitmap {
            // 이미지의 확대/축소 비율을 구한다.
            val ratio = targetWidth.toDouble() / bitmap.width.toDouble()
            // 세로 길이를 구한다.
            val targetHeight = (bitmap.height * ratio).toInt()
            // 크기를 조장한 Bitmap을 생성한다.
            val resizedBitmap = Bitmap.createScaledBitmap(bitmap, targetWidth, targetHeight, false)

            return resizedBitmap
        }
```

카메라 액티비티를 실행하기 위한 런처와 사진이 저장될 경로를 가지고 있을 Uri 를 정의해준다.

```
[AddContentFragment.kt]

    // Activity 실행을 위한 런처
    lateinit var cameraLauncher: ActivityResultLauncher<Intent>

    // 촬영된 사진이 저장된 경로 정보를 가지고 있는 Uri 객체
    lateinit var contentUri: Uri
```

케마라 액티비티리를 실행하는 메서드를 만들어준다.

```
[AddContentFragment.kt]

    // 카메라 런처를 실행하는 메서드
    fun startCameraLauncher(){
        // 촬영한 사진이 저장될 경로를 가져온다.
        contentUri = Tools.getPictureUri(contentActivity, "kr.co.lion.androidproject4boardapp.file_provider")

        if(contentUri != null){
            // 실행할 액티비티를 카메라 액티비티로 지정한다.
            // 단말기에 설치되어 있는 모든 애플리케이션이 가진 액티비티 중에 사진촬영이
            // 가능한 액티비가 실행된다.
            val cameraIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
            // 이미지가 저장될 경로를 가지고 있는 Uri 객체를 인텐트에 담아준다.
            cameraIntent.putExtra(MediaStore.EXTRA_OUTPUT, contentUri)
            // 카메라 액티비티 실행
            cameraLauncher.launch(cameraIntent)
        }
    }
```

카메라 런처를 만들어주는 메서드를 구현한다.

```
[AddContentFragment.kt]

    // 카메라 런처 설정
    fun settingCameraLauncher(){
        val contract1 = ActivityResultContracts.StartActivityForResult()
        cameraLauncher = registerForActivityResult(contract1){
            // 사진을 사용하겠다고 한 다음에 돌아왔을 경우
            if(it.resultCode == AppCompatActivity.RESULT_OK){
                // 사진 객체를 생성한다.
                val bitmap = BitmapFactory.decodeFile(contentUri.path)

                // 회전 각도값을 구한다.
                val degree = Tools.getDegree(contentActivity, contentUri)
                // 회전된 이미지를 구한다.
                val bitmap2 = Tools.rotateBitmap(bitmap, degree.toFloat())
                // 크기를 조정한 이미지를 구한다.
                val bitmap3 = Tools.resizeBitmap(bitmap2, 1024)

                fragmentAddContentBinding.imageViewAddContent.setImageBitmap(bitmap3)

                // 사진 파일을 삭제한다.
                val file = File(contentUri.path)
                file.delete()
            }
        }
    }
```

카메라 런처를 생성하는 메서드를 호출해준다.

```
[AddContentFragment - onCreateView]

        settingCameraLauncher()
```

카메라 런처를 실행하는 메서드를 호출해준다.

```
[AddContentFragment - settingToolbarAddContent]

                        // 카메라
                        R.id.menuItemAddContentCamera -> {
                            startCameraLauncher()
                        }
```

020 앨범 관련 작업

권한을 등록해준다.

```
[AndroidManifest.xml]

    <!-- 앨범으로 부터 사진을 가져오기 위한 권한 -->
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.ACCESS_MEDIA_LOCATION"/>
```

확인할 권한 목록을 작성해준다.

```
[MainActivity.kt]

    // 확인할 권한 목록
    val permissionList = arrayOf(
        Manifest.permission.READ_EXTERNAL_STORAGE,
        Manifest.permission.ACCESS_MEDIA_LOCATION
    )
```

권한을 확인하는 메서드를 호출한다.

```
{MainActivity.kt - onCreate}

        // 권한 확인
        requestPermissions(permissionList, 0)
```

앨범 런처를 담을 프로퍼티를 정의한다.

[AddContentFragment.kt]

```
    lateinit var albumLauncher:ActivityResultLauncher<Intent>
```

앨범 런처를 설정하는 메서드를 구현해준다.

[AddContentFragmeht.kt]

```
    // 앨범 런처 설정
    fun settingAlbumLauncher() {
        // 앨범 실행을 위한 런처
        val contract2 = ActivityResultContracts.StartActivityForResult()
        albumLauncher = registerForActivityResult(contract2){
            // 사진 선택을 완료한 후 돌아왔다면
            if(it.resultCode == AppCompatActivity.RESULT_OK){
                // 선택한 이미지의 경로 데이터를 관리하는 Uri 객체를 추출한다.
                val uri = it.data?.data
                if(uri != null){
                    // 안드로이드 Q(10) 이상이라면
                    val bitmap = if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q){
                        // 이미지를 생성할 수 있는 객체를 생성한다.
                        val source = ImageDecoder.createSource(contentActivity.contentResolver, uri)
                        // Bitmap을 생성한다.
                        ImageDecoder.decodeBitmap(source)
                    } else {
                        // 컨텐츠 프로바이더를 통해 이미지 데이터에 접근한다.
                        val cursor = contentActivity.contentResolver.query(uri, null, null, null, null)
                        if(cursor != null){
                            cursor.moveToNext()

                            // 이미지의 경로를 가져온다.
                            val idx = cursor.getColumnIndex(MediaStore.Images.Media.DATA)
                            val source = cursor.getString(idx)

                            // 이미지를 생성한다
                            BitmapFactory.decodeFile(source)
                        }  else {
                            null
                        }
                    }

                    // 회전 각도값을 가져온다.
                    val degree = Tools.getDegree(contentActivity, uri)
```

```
                    val degree = Tools.getDegree(contentActivity, uri)
                    // 회전 이미지를 가져온다
                    val bitmap2 = Tools.rotateBitmap(bitmap!!, degree.toFloat())
                    // 크기를 줄인 이미지를 가져온다.
                    val bitmap3 = Tools.resizeBitmap(bitmap2, 1024)

                    fragmentAddContentBinding.imageViewAddContent.setImageBitmap(bitmap3)
                }
            }
        }
    }
```

메서드를 호출한다.

```
[AddContentFragment - onCreateview]

settingAlbumLauncher()
```

앨범 런처를 실행하는 메서드를 구현한다.

```
    // 앨범 런처를 실행하는 메서드
    fun startAlbumLauncher(){
        // 앨범에서 사진을 선택할 수 있도록 셋팅된 인텐트를 생성한다.
        val albumIntent = Intent(Intent.ACTION_PICK, MediaStore.Images.Media.EXTERNAL_CONTENT_URI)
        // 실행할 액티비티의 타입을 설정(이미지를 선택할 수 있는 것이 뜨게 한다)
        albumIntent.setType("image/*")
        // 선택할 수 있는 파들의 MimeType을 설정한다.
        // 여기서 선택한 종류의 파일만 선택이 가능하다. 모든 이미지로 설정한다.
        val mimeType = arrayOf("image/*")
        albumIntent.putExtra(Intent.EXTRA_MIME_TYPES, mimeType)
        // 액티비티를 실행한다.
        albumLauncher.launch(albumIntent)
    }
```

메뉴를 눌렀을 때 메서드를 호출한다.

[AddContentFragmeht.kt - settingToolbarAddcontent]

```
                        // 앨범
                        R.id.menuItemModifyContentAlbum -> {
                            startAlbumLauncher()
                        }
```

초기화 메뉴를 눌렀을 때 초기화 메서드를 호출한다.

[AddcontentFragmeht.kt - settingToolbarAddContent]

```
                        // 초기화
                        R.id.menuItemModifyContentReset -> {
                            settingInputForm()
                        }
```

021 글작성 처리 작업

입력 유효성 검사 메서드를 작성해준다.

[AddContentFragment.kt]

```
    // 입력 요소 유효성 검사
    fun checkInputForm():Boolean{
        // 입력한 내용을 가져온다.
        val subject = addContentViewModel.textFieldAddContentSubject.value!!
        val text = addContentViewModel.textFieldAddContentText.value!!

        if(subject.isEmpty()){
            Tools.showErrorDialog(contentActivity, fragmentAddContentBinding.textFieldAddContentSubject,
                "제목 입력 오류", "제목을 입력해주세요")
            return false
        }

        if(text.isEmpty()){
            Tools.showErrorDialog(contentActivity, fragmentAddContentBinding.textFieldAddContentText,
                "내용 입력 오류", "내용을 입력해주세요")
            return false
        }

        return true
    }
```

유효성 검새 메서드를 호출해준다.

[AddContentFragment.kt - settingToolbarAddContent]

```
                        // 완료
                        R.id.menuItemModifyContentDone -> {
                            // 입력 요소 유효성 검사
                            val chk = checkInputForm()

                            if(chk == true) {
                                // ReadContentFragment로 이동한다.
                                contentActivity.replaceFragment(ContentFragmentName.READ_CONTENT_FRAGMENT, true, true, null)
                            }
                        }
```

022 글 읽는 화면 작업

ReadContentViewModel 클래스 생성

```
[ReadContentViewModel.kt]

class ReadContentViewModel : ViewModel(){

    // 제목
    val textFieldReadContentSubject = MutableLiveData<String>()
    // 게시판 종류
    val textFieldReadContentType = MutableLiveData<String>()
    // 닉네임
    val textFieldReadContentNickName = MutableLiveData<String>()
    // 작성 날짜
    val textFieldReadContentDate = MutableLiveData<String>()
    // 글 내용
    val textFieldReadContentText = MutableLiveData<String>()
}
```

layout 파일에 ViewModel 객체를 정의해준다.

```
[fragment_read_content.xml]

    <data>
        <variable
            name="readContentViewModel"
            type="kr.co.lion.androidproject4boardapp.viewmodel.ReadContentViewModel" />
    </data>
```

layout 파일의 입력 요소에 MutableLiveData 를 연결해준다.

```
[fragment_read_content.xml]

android:text="@{readContentViewModel.textFieldReadContentSubject}"
android:text="@{readContentViewModel.textFieldReadContentType}"
android:text="@{readContentViewModel.textFieldReadContentNickName}"
android:text="@{readContentViewModel.textFieldReadContentDate}"
android:text="@{readContentViewModel.textFieldReadContentText}"
```

ViewModel 기본 코드를 작성해준다.

```
[ReadContentFragment.kt]

    lateinit var readContentViewModel: ReadContentViewModel

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {
        // Inflate the layout for this fragment

        // fragmentReadContentBinding = FragmentReadContentBinding.inflate(inflater)
        fragmentReadContentBinding = DataBindingUtil.inflate(inflater, R.layout.fragment_read_content, container, false)
        readContentViewModel = ReadContentViewModel()
        fragmentReadContentBinding.readContentViewModel = readContentViewModel
        fragmentReadContentBinding.lifecycleOwner = this
```

글 내용을 설정해준다.

```
[ReadContentFragment.kt]

    // 입력 요소에 값을 넣어준다.
    fun settingInputForm(){
        readContentViewModel.textFieldReadContentSubject.value = "제목입니다"
        readContentViewModel.textFieldReadContentType.value = "자유게시판"
        readContentViewModel.textFieldReadContentNickName.value = "홍길동"
        readContentViewModel.textFieldReadContentDate.value = "2024-03-12"
        readContentViewModel.textFieldReadContentText.value ="내용입니다"
    }
```

메서드를 호출해준다.

```
[ReadContentFragment.kt - onCreateView]


settingInputForm()
```

023 글 수정 화면 작업

ModifyContentViewModel 클래스를 생성한다.

```
[ModifyContentViewModel.kt]
```

```
package kr.co.lion.androidproject4boardapp.viewmodel

import androidx.databinding.BindingAdapter
import androidx.databinding.InverseBindingAdapter
import androidx.databinding.InverseBindingListener
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import com.google.android.material.button.MaterialButtonToggleGroup
import kr.co.lion.androidproject4boardapp.ContentType
import kr.co.lion.androidproject4boardapp.R

class ModifyContentViewModel : ViewModel() {
    // 글 제목
    val textFieldModifyContentSubject = MutableLiveData<String>()
    // 게시판 종류
    val toggleModifyContentType = MutableLiveData<Int>()
    // 글 내용
    val textFieldModifyContentText = MutableLiveData<String>()

    // 게시판 종류를 받아 MutableLiveData에 설정하는 메서드
    fun settingContentType(contentType: ContentType){
        toggleModifyContentType.value = when(contentType){
            ContentType.TYPE_ALL -> 0
            ContentType.TYPE_FREE -> R.id.buttonModifyContentType1
            ContentType.TYPE_HUMOR -> R.id.buttonModifyContentType2
            ContentType.TYPE_SOCIETY -> R.id.buttonModifyContentType3
            ContentType.TYPE_SPORTS -> R.id.buttonModifyContentType4
        }
    }
    companion object {
        // toggleAddContentType 에 값을 설정했을 때 checkedButtonId 속성에 값을 반영해 줄 때 호출(순방향)
        @BindingAdapter("android:checkedButtonId")
        @JvmStatic
        fun setCheckedButtonId(group: MaterialButtonToggleGroup, buttonId:Int){
            group.check(buttonId)
        }

        // 안드로이드 OS가 현재 화면을 구성하고자 할 때 InverseBindingAdapter에 등록되어 있는 event 값을 보고
        // event에 등록되어 있는 이름과 동일한 BindingAdapter를 찾아 메서드를 호출해준다.
        // 리스너 설정을 해준다.
        @BindingAdapter("checkedButtonChangeListener")
        @JvmStatic
        fun checkedButtonChangeListener(group: MaterialButtonToggleGroup, inverseBindingListener: InverseBindingListener){
```

```
            group.addOnButtonCheckedListener { group, checkedId, isChecked ->
                inverseBindingListener.onChange()
            }
        }

        // 안드로이드 OS가 현재 화면을 구성하고자 할 때 속성에 설정된 MutableLiveData가 양방향(@=)으로 되어 있을 경우
        // 참고할 데이터가 셋팅되어 있는 메서드
        // inverseBindingListener.onChange() 호출시 동작할 메서드
        @InverseBindingAdapter(attribute = "android:checkedButtonId", event="checkedButtonChangeListener")
        @JvmStatic
        fun getCheckedButtonId(group: MaterialButtonToggleGroup):Int{
            return group.checkedButtonId
        }
    }
}
```

layout 파일에 ViewModel 설정을 해준다.

```
[fragment_modify_content.xml]

    <data>
        <variable
            name="modifyContentViewModel"
            type="kr.co.lion.androidproject4boardapp.viewmodel.ModifyContentViewModel" />
    </data>

    android:text="@={modifyContentViewModel.textFieldModifyContentSubject}"
    android:checkedButtonId="@={modifyContentViewModel.toggleModifyContentType}"
    android:text="@={modifyContentViewModel.textFieldModifyContentText}"
```

ViewModel 기본 코드를 넣어준다.

```
[ModifyContentFragment.kt]

    lateinit var modifyContentViewModel: ModifyContentViewModel

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {
        // Inflate the layout for this fragment

        // fragmentModifyContentBinding = FragmentModifyContentBinding.inflate(inflater)
        fragmentModifyContentBinding = DataBindingUtil.inflate(inflater, R.layout.fragment_modify_content, container, false)
        modifyContentViewModel = ModifyContentViewModel()
        fragmentModifyContentBinding.modifyContentViewModel = modifyContentViewModel
        fragmentModifyContentBinding.lifecycleOwner = this
```

메서드를 호출해준다.

```
[ModifyCotnentFragment.kt - onCreateView]

        settingInputForm()
```

Activity 런처와 촬영된 사진을 저장할 경로를 가진 Uri를 선언한다.

```
[ModifyCotnentFragment.kt]

    // Activity 실행을 위한 런처
    lateinit var cameraLauncher: ActivityResultLauncher<Intent>
    lateinit var albumLauncher:ActivityResultLauncher<Intent>

    // 촬영된 사진이 저장된 경로 정보를 가지고 있는 Uri 객체
    lateinit var contentUri: Uri
```

카메라 런처 설정, 카메라 런처 실행, 앨범 런처 설정, 앨범 런처 실행 메서드를 구현해준다

```
    // 카메라 런처 설정
    fun settingCameraLauncher(){
        val contract1 = ActivityResultContracts.StartActivityForResult()
        cameraLauncher = registerForActivityResult(contract1){
            // 사진을 사용하겠다고 한 다음에 돌아왔을 경우
            if(it.resultCode == AppCompatActivity.RESULT_OK){
                // 사진 객체를 생성한다
```

```
                // 사진 객체를 생성한다.
                val bitmap = BitmapFactory.decodeFile(contentUri.path)

                // 회전 각도값을 구한다.
                val degree = Tools.getDegree(contentActivity, contentUri)
                // 회전된 이미지를 구한다.
                val bitmap2 = Tools.rotateBitmap(bitmap, degree.toFloat())
                // 크기를 조정한 이미지를 구한다.
                val bitmap3 = Tools.resizeBitmap(bitmap2, 1024)

                fragmentModifyContentBinding.imageViewModifyContent.setImageBitmap(bitmap3)

                // 사진 파일을 삭제한다.
                val file = File(contentUri.path)
                file.delete()
            }
        }
    }

    // 카메라 런처를 실행하는 메서드
    fun startCameraLauncher(){
        // 촬영한 사진이 저장될 경로를 가져온다.
        contentUri = Tools.getPictureUri(contentActivity, "kr.co.lion.androidproject4boardapp.file_provider")

        if(contentUri != null){
            // 실행할 액티비티를 카메라 액티비티로 지정한다.
            // 단말기에 설치되어 있는 모든 애플리케이션이 가진 액티비티 중에 사진촬영이
            // 가능한 액티비가 실행된다.
            val cameraIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
            // 이미지가 저장될 경로를 가지고 있는 Uri 객체를 인텐트에 담아준다.
            cameraIntent.putExtra(MediaStore.EXTRA_OUTPUT, contentUri)
            // 카메라 액티비티 실행
            cameraLauncher.launch(cameraIntent)
        }
    }

    // 앨범 런처 설정
    fun settingAlbumLauncher() {
        // 앨범 실행을 위한 런처
        val contract2 = ActivityResultContracts.StartActivityForResult()
        albumLauncher = registerForActivityResult(contract2){
            // 사진 선택을 완료한 후 돌아왔다면
            if(it.resultCode == AppCompatActivity.RESULT_OK){
                // 선택한 이미지의 경로 데이터를 관리하는 Uri 객체를 추출한다.
                val uri = it.data?.data
```

```
                val uri = it.data?.data
                if(uri != null){
                    // 안드로이드 Q(10) 이상이라면
                    val bitmap = if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q){
                        // 이미지를 생성할 수 있는 객체를 생성한다.
                        val source = ImageDecoder.createSource(contentActivity.contentResolver, uri)
                        // Bitmap을 생성한다.
                        ImageDecoder.decodeBitmap(source)
                    } else {
                        // 컨텐츠 프로바이더를 통해 이미지 데이터에 접근한다.
                        val cursor = contentActivity.contentResolver.query(uri, null, null, null, null)
                        if(cursor != null){
                            cursor.moveToNext()

                            // 이미지의 경로를 가져온다.
                            val idx = cursor.getColumnIndex(MediaStore.Images.Media.DATA)
                            val source = cursor.getString(idx)

                            // 이미지를 생성한다
                            BitmapFactory.decodeFile(source)
                        }  else {
                            null
                        }
                    }

                    // 회전 각도값을 가져온다.
                    val degree = Tools.getDegree(contentActivity, uri)
                    // 회전 이미지를 가져온다
                    val bitmap2 = Tools.rotateBitmap(bitmap!!, degree.toFloat())
                    // 크기를 줄인 이미지를 가져온다.
                    val bitmap3 = Tools.resizeBitmap(bitmap2, 1024)

                    fragmentModifyContentBinding.imageViewModifyContent.setImageBitmap(bitmap3)
                }
            }
        }
    }

    // 앨범 런처를 실행하는 메서드
    fun startAlbumLauncher(){
        // 앨범에서 사진을 선택할 수 있도록 셋팅된 인텐트를 생성한다.
        val albumIntent = Intent(Intent.ACTION_PICK, MediaStore.Images.Media.EXTERNAL_CONTENT_URI)
        // 실행할 액티비티의 타입을 설정(이미지를 선택할 수 있는 것이 뜨게 한다)
        albumIntent.setType("image/*")
        // 선택할 수 있는 파들의 MimeType을 설정한다.
```

```
        // 여기서 선택한 종류의 파일만 선택이 가능하다. 모든 이미지로 설정한다.
        val mimeType = arrayOf("image/*")
        albumIntent.putExtra(Intent.EXTRA_MIME_TYPES, mimeType)
        // 액티비티를 실행한다.
        albumLauncher.launch(albumIntent)
    }
```

카메라 런처 설정, 앨범 런처 설정 메서드를 호출해준다.

```
[ModifyContentFragment.kt - onCreateView]

        settingCameraLauncher()
        settingAlbumLauncher()
```

메뉴를 누르면 카메라 런처와 앨범 런처가 실행되도록 메서드를 호출해준다.

```
                        // 카메라
                        R.id.menuItemModifyContentCamera -> {
                            startCameraLauncher()
                        }
                        // 앨범
                        R.id.menuItemModifyContentAlbum -> {
                            startAlbumLauncher()
                        }
```

초기화 메뉴를 눌렀을 때 초기화 메서드를 호출해준다.

```
[ModifyContentFragment.kt]

                        //  초기화
                        R.id.menuItemModifyContentReset -> {
                            settingInputForm()
                        }
```

입력 요소에 대한 검사 메서드를 구현한다.

[ModifyContentFragment.kt]

```
    // 입력 요소의 유효성 검사
    fun checkInputForm():Boolean{
        // 입력한 내용을 가져온다.
        val subject = modifyContentViewModel.textFieldModifyContentSubject.value!!
        val text = modifyContentViewModel.textFieldModifyContentText.value!!

        if(subject.isEmpty()){
            Tools.showErrorDialog(contentActivity, fragmentModifyContentBinding.textFieldModifyContentSubject,
                "제목 입력 오류", "제목을 입력해주세요")
            return false
        }

        if(text.isEmpty()){
            Tools.showErrorDialog(contentActivity, fragmentModifyContentBinding.textFieldModifyContentText,
                "내용 입력 오류", "내용을 입력해주세요")
            return false
        }

        return true
    }
```

완료 메뉴를 누르면 유효성 검사 후 이전으로 돌아가게 한다.

[ModifyContentFragment.kt - settingToolbarModifyContent]

```
                        // 완료
                        R.id.menuItemModifyContentDone -> {
                            // 입력 요소 검사
                            val chk = checkInputForm()

                            if(chk == true){
                                contentActivity.removeFragment(ContentFragmentName.MODIFY_CONTENT_FRAGMENT)
                            }
                        }
```

024 사용자 정보 수정 화면 작업

ModifyUserViewModel 클래스를 생성한다.

```
package kr.co.lion.androidproject4boardapp.viewmodel

import androidx.databinding.BindingAdapter
import androidx.databinding.InverseBindingAdapter
import androidx.databinding.InverseBindingListener
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import com.google.android.material.button.MaterialButtonToggleGroup
import com.google.android.material.checkbox.MaterialCheckBox
import kr.co.lion.androidproject4boardapp.Gender
import kr.co.lion.androidproject4boardapp.R

class ModifyUserViewModel : ViewModel(){
    // 닉네임
    val textFieldModifyUserInfoNickName = MutableLiveData<String>()
    // 나이
    val textFieldModifyUserInfoAge = MutableLiveData<String>()
    // 비밀번호
    val textFieldModifyUserPw = MutableLiveData<String>()
    // 비멀번호 확인
    val textFieldModifyUserPw2 = MutableLiveData<String>()
    // 성별
    val toggleModifyUserInfoGender = MutableLiveData<Int>()

    // 취미들
    val checkBoxModifyUserInfoHobby1 = MutableLiveData<Boolean>()
    val checkBoxModifyUserInfoHobby2 = MutableLiveData<Boolean>()
    val checkBoxModifyUserInfoHobby3 = MutableLiveData<Boolean>()
    val checkBoxModifyUserInfoHobby4 = MutableLiveData<Boolean>()
    val checkBoxModifyUserInfoHobby5 = MutableLiveData<Boolean>()
    val checkBoxModifyUserInfoHobby6 = MutableLiveData<Boolean>()

    // 취미 전체
    val checkBoxModifyUserInfoAllState = MutableLiveData<Int>()
    val checkBoxModifyUserInfoAll = MutableLiveData<Boolean>()

    // 성별을 셋팅하는 메서드
    fun settingGender(gender: Gender){
        // 성별로 분기한다.
        when(gender){
            Gender.MALE -> {
```

```
                toggleModifyUserInfoGender.value = R.id.buttonModifyUserInfoMale
            }
            Gender.FEMALE -> {
                toggleModifyUserInfoGender.value = R.id.buttonModifyUserInfoFemale
            }
        }
    }

    companion object {
        // ViewModel에 값을 설정하여 화면에 반영하는 작업을 할 때 호출된다.
        // () 안에는 속성의 이름을 넣어준다. 속성의 이름은 자유롭게 해주면 되지만 기존의 속성 이름과 중복되지 않게
        // 해줘야 한다.
        // 매개변수 : 값이 설정된 View 객체, ViewModel을 통해 설정되는 값
        @BindingAdapter("android:checkedButtonId")
        @JvmStatic
        fun setCheckedButtonId(group: MaterialButtonToggleGroup, buttonId:Int){
            group.check(buttonId)
        }

        // 값을 속성에 넣어주는 것을 순방향이라고 부른다. 반대로 속성의 값이 변경되어 MutableLive 데이터로
        // 전달하는 것을 역방햐이라고 한다.
        // 순방향만 구현해주면 단방향이 되는 것이고 순방향과 역방향을 모두 구현해주면 양방향
        // 화면 요소가 가진 속성애 새로운 값이 설정되면 ViewModel의 변수에 값이 설정될 때 호출된다.
        // 리스너 역할을 할 속성을 만들어준다.
        @BindingAdapter("checkedButtonChangeListener")
        @JvmStatic
        fun checkedButtonChangeListener(group: MaterialButtonToggleGroup, inverseBindingListener: InverseBindingListener){
            group.addOnButtonCheckedListener { group, checkedId, isChecked ->
                inverseBindingListener.onChange()
            }
        }
        // 역방향 바인딩이 벌어질 때 호출된다.
        @InverseBindingAdapter(attribute = "android:checkedButtonId", event="checkedButtonChangeListener")
        @JvmStatic
        fun getCheckedButtonId(group: MaterialButtonToggleGroup):Int{
            return group.checkedButtonId
        }
    }

    // 체크박스 전체 상태를 설정하는 메서드
    fun setCheckAll(checked:Boolean){
        checkBoxModifyUserInfoHobby1.value = checked
        checkBoxModifyUserInfoHobby2.value = checked
        checkBoxModifyUserInfoHobby3.value = checked
```

```
        checkBoxModifyUserInfoHobby4.value = checked
        checkBoxModifyUserInfoHobby5.value = checked
        checkBoxModifyUserInfoHobby6.value = checked
    }

    // 전체 취미 체크박스를 누르면...
    fun onCheckBoxAllChanged(){
        // 전체 취미 체크박스의 체크 여부를 모든 체크박스에 설정해준다.
        setCheckAll(checkBoxModifyUserInfoAll.value!!)
    }

    // 각 체크박스를 누르면...
    fun onCheckBoxChanged(){
        // 체크되어 있는 체크박스 개수를 담을 변수
        var checkedCnt = 0

        // 체크되어 있다면 체크되어 있는 체크박스의 개수를 1 증가시킨다.
        if(checkBoxModifyUserInfoHobby1.value == true){
            checkedCnt++
        }
        if(checkBoxModifyUserInfoHobby2.value == true){
            checkedCnt++
        }
        if(checkBoxModifyUserInfoHobby3.value == true){
            checkedCnt++
        }
        if(checkBoxModifyUserInfoHobby4.value == true){
            checkedCnt++
        }
        if(checkBoxModifyUserInfoHobby5.value == true){
            checkedCnt++
        }
        if(checkBoxModifyUserInfoHobby6.value == true){
            checkedCnt++
        }

        // 체크되어 있는 것이 없다면
        if(checkedCnt == 0){
            checkBoxModifyUserInfoAll.value = false
            checkBoxModifyUserInfoAllState.value = MaterialCheckBox.STATE_UNCHECKED
        }
        // 모두 체크되어 있다면
        else if(checkedCnt == 6){
            checkBoxModifyUserInfoAll.value = true
            checkBoxModifyUserInfoAllState.value = MaterialCheckBox.STATE_CHECKED
```

```
            checkBoxModifyUserInfoAllState.value = MaterialCheckBox.STATE_CHECKED
        }
        // 일부만 체크되어 있다면
        else {
            checkBoxModifyUserInfoAllState.value = MaterialCheckBox.STATE_INDETERMINATE
        }
    }

}
```

layout파일에 ViewModel 설정을 해준다.

```
[res/layout/fragment_modify_user.xml]

    <data>
        <variable
            name="modifyUserViewModel"
            type="kr.co.lion.androidproject4boardapp.viewmodel.ModifyUserViewModel" />
    </data>

    android:text="@={modifyUserViewModel.textFieldModifyUserInfoNickName}"

    android:text="@={modifyUserViewModel.textFieldModifyUserInfoAge}"

    android:text="@={modifyUserViewModel.textFieldModifyUserPw}"

    android:text="@={modifyUserViewModel.textFieldModifyUserPw2}"

    android:checkedButtonId="@={modifyUserViewModel.toggleModifyUserInfoGender}"

    app:checkedState="@{modifyUserViewModel.checkBoxModifyUserInfoAllState}"
    android:checked="@={modifyUserViewModel.checkBoxModifyUserInfoAll}"
    android:onClickListener="@{ (view) -> modifyUserViewModel.onCheckBoxAllChanged()}"

    android:checked="@={modifyUserViewModel.checkBoxModifyUserInfoHobby1}"
    android:onClickListener="@{ (view) -> modifyUserViewModel.onCheckBoxChanged()}"

    android:checked="@={modifyUserViewModel.checkBoxModifyUserInfoHobby2}"
    android:onClickListener="@{ (view) -> modifyUserViewModel.onCheckBoxChanged()}"

    android:checked="@={modifyUserViewModel.checkBoxModifyUserInfoHobby3}"
    android:onClickListener="@{ (view) -> modifyUserViewModel.onCheckBoxChanged()}"

    android:checked="@={modifyUserViewModel.checkBoxModifyUserInfoHobby4}"
    android:onClickListener="@{ (view) -> modifyUserViewModel.onCheckBoxChanged()}"

    android:checked="@={modifyUserViewModel.checkBoxModifyUserInfoHobby5}"
    android:onClickListener="@{ (view) -> modifyUserViewModel.onCheckBoxChanged()}"

    android:checked="@={modifyUserViewModel.checkBoxModifyUserInfoHobby6}"
    android:onClickListener="@{ (view) -> modifyUserViewModel.onCheckBoxChanged()}"
```

Fragment에 ViewModel 기본 코드를 넣어준다

```
[ModifyUserFragment.kt]

    lateinit var modifyUserViewModel: ModifyUserViewModel

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {
        // Inflate the layout for this fragment

        // fragmentModifyUserBinding = FragmentModifyUserBinding.inflate(inflater)
        fragmentModifyUserBinding = DataBindingUtil.inflate(inflater, R.layout.fragment_modify_user, container, false)
        modifyUserViewModel = ModifyUserViewModel()
        fragmentModifyUserBinding.modifyUserViewModel = modifyUserViewModel
        fragmentModifyUserBinding.lifecycleOwner = this
```

입력 요소 초기화 메서드를 구현한다

```
[ModifyUserFragment.kt]

    // 입력 요소 초기화
    fun settingInputForm(){
        modifyUserViewModel.textFieldModifyUserInfoNickName.value = "홍길동"
        modifyUserViewModel.textFieldModifyUserInfoAge.value = "100"
        modifyUserViewModel.textFieldModifyUserPw.value = ""
        modifyUserViewModel.textFieldModifyUserPw2.value = ""
        modifyUserViewModel.settingGender(Gender.FEMALE)

        modifyUserViewModel.checkBoxModifyUserInfoHobby1.value = true
        modifyUserViewModel.checkBoxModifyUserInfoHobby2.value = true
        modifyUserViewModel.checkBoxModifyUserInfoHobby3.value = false
        modifyUserViewModel.checkBoxModifyUserInfoHobby4.value = true
        modifyUserViewModel.checkBoxModifyUserInfoHobby5.value = true
        modifyUserViewModel.checkBoxModifyUserInfoHobby6.value = false

        modifyUserViewModel.onCheckBoxChanged()
    }
```

메서드를 호출한다.

```
[ModifyUserFramgent.kt - onCreateView ]

settingInputForm()
```

초기화 메뉴를 추가한다.

```
[res/menu/menu_modify_user.xml]

    <item
        android:id="@+id/menuItemModifyUserReset"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:icon="@drawable/clear_all_24px"
        android:title="초기화"
        app:showAsAction="always" />
```

메뉴를 눌렀을 때 동작하는 부분을 구현해준다.

```
[ModifyUserFragament.kt - settingToolbarModifyuser]

                setOnMenuItemClickListener {

                    when(it.itemId){
                        // 초기화
                        R.id.menuItemModifyUserReset ->{
                            settingInputForm()
                        }
                        // 완료
                        R.id.menuItemModifyUserDone ->{

                        }
                    }
                    true
                }
```

입력 요소에 대한 유효성 검사 코드를 넣어준다.

[ModifyUserFragment.kt]

```
    // 입력 요소에 대한 유효성 검사
    fun checkInputForm():Boolean{
        // 입력 요소 값들을 가져온다.
        val nickName = modifyUserViewModel.textFieldModifyUserInfoNickName.value!!
        val age = modifyUserViewModel.textFieldModifyUserInfoAge.value!!
        val pw = modifyUserViewModel.textFieldModifyUserPw.value!!
        val pw2 = modifyUserViewModel.textFieldModifyUserPw2.value!!

        if(nickName.isEmpty()){
            Tools.showErrorDialog(contentActivity, fragmentModifyUserBinding.textFieldModifyUserInfoNickName,
                "닉네임 입력 오류", "닉네임을 입력해주세요")
            return false
        }

        if(age.isEmpty()){
            Tools.showErrorDialog(contentActivity, fragmentModifyUserBinding.textFieldModifyUserInfoAge,
                "나이 입력 오류", "나이를 입력해주세요")
            return false
        }

        // 비밀번호 둘 중 하나라도 비어있지 않고 서로 다르다면....
        if((pw.isNotEmpty() || pw2.isNotEmpty()) && pw != pw2){

            modifyUserViewModel.textFieldModifyUserPw.value = ""
            modifyUserViewModel.textFieldModifyUserPw2.value = ""

            Tools.showErrorDialog(contentActivity, fragmentModifyUserBinding.textFieldModifyUserPw,
                "비밀번호 입력 오류", "비밀번호가 다릅니다.")
            return false
        }

        return true
    }
```

완료 메뉴를 눌렀을 때를 처리해준다.

```
[ModifyUserFragment.kt - settingToolbarModifyUser]

                        // 완료
                        R.id.menuItemModifyUserDone ->{
                            // 유효성 감사를 한다.
                            val chk = checkInputForm()

                            if(chk == true){
                                Tools.hideSoftInput(contentActivity)
                            }
                        }
```

네비게이션 설정이 빠져있어 이를 구현해준다.

```
[ModifyUserFragment.kt - settingToolbarModifyUser]

                // 네비게이션 아이콘 설정
                setNavigationIcon(R.drawable.menu_24px)
                setNavigationOnClickListener {
                    contentActivity.activityContentBinding.drawerLayoutContent.open()
                }
```

FireBase 서비스 설정

Android Project PPT 57, 58, 61 참고

025 회원 가입 처리

Firebase store 에 들어가서 다음 컬렉션을 만들어준다.

Sequence - UserSequence - value : 0

다음으로 이동 처리하는 메서드를 만들어준다.

```
[JoinFragment.kt]

    // 다음 과정으로 이동한다
    fun joinNext(){
        // 사용자가 입력한 데이터를 담는다
        val joinBundle = Bundle()
        joinBundle.putString("joinUserId", joinViewModel.textFieldJoinUserId.value!!)
        joinBundle.putString("joinUserPw", joinViewModel.textFieldJoinUserPw.value!!)

        // 키보를 내려준다.
        Tools.hideSoftInput(mainActivity)
        // AddUserInfoFragment를 보여준다.
        mainActivity.replaceFragment(MainFragmentName.ADD_USER_INFO_FRAGMENT, true, true, joinBundle)
    }
```

다음 버튼을 누르면 동작하는 코드를 삭제하고 메서드를 호출해준다.

```
[JoinFragment.kt - settingButtonJoinNext]

                    // 입력이 모두 잘 되어 있다면..
                    if(chk == true) {
//                      // 키보를 내려준다.
//                      Tools.hideSoftInput(mainActivity)
//                      // AddUserInfoFragment를 보여준다.
//                      mainActivity.replaceFragment(MainFragmentName.ADD_USER_INFO_FRAGMENT, true, true, null)
                        joinNext()
                    }
```

model 패키지를 만들어주고 회원 정보를 관리할 클래스를 만들어준다.

```
[UserModel.kt]

// 회원번호, 아이디, 비밀번호, 닉네임, 나이, 성별 , 취미6가지, 회원상태
data class UserModel(var userIdx:Int, var userId:String,
    var userPw:String, var userNickName:String, var userAge:Int, var userGender:Int,
    var userHobby1:Boolean, var userHobby2:Boolean, var userHobby3: Boolean,
    var userHobby4: Boolean, var userHobby5: Boolean, var userHobby6: Boolean,
    var userState:Int)
```

성별을 나타내는 값에 숫자 값을 추가해주고 회원 상태값을 정의해준다.

```
[Tools.kt]


// 남자 또는 여자를 나타내는 값을 정의한다.
enum class Gender(var str:String, var num:Int){
    MALE("male", 1),
    FEMALE("female", 2)
}

// 회원 상태를 나타내는 값을 정의한다
enum class UserState(var str:String, var num:Int){
    USER_STATE_NORMAL("정상", 1),
    USER_STATE_SIGNOUT("탈퇴", 2),
}
```

사용자 번호 시퀀스를 가져오는 메서드를 만들어준다.

```
[UserDao.kt]

        // 사용자 번호 시퀀스값을 가져온다.
        suspend fun getUserSequence():Int{

            var userSequence = 0

            val job1 = CoroutineScope(Dispatchers.IO).launch {
                // 컬렉션에 접근할 수 있는 객체를 가져온다.
                val collectionReference = Firebase.firestore.collection("Sequence")
                // 사용자 번호 시퀀스 값을 가지고 있는 문서에 접근할 수 있는 객체를 가져온다.
                val documentReference = collectionReference.document("UserSequence")
                // 문서내에 있는 데이터를 가져올 수 있는 객체를 가져온다.
                val documentSnapShot = documentReference.get().await()
                userSequence = documentSnapShot.getLong("value")?.toInt()!!
            }
            job1.join()

            return userSequence
        }
```

사용자 번호 시퀀스를 업데이트 하는 메서드를 만들어준다.

[UserDao.kt]

```
        // 사용자 시퀀스 값을 업데이트 한다.
        suspend fun updateUserSequence(userSequence:Int){
            val job1 = CoroutineScope(Dispatchers.IO).launch {
                // 컬렉션에 접근할 수 있는 객체를 가져온다.
                val collectionReference = Firebase.firestore.collection("Sequence")
                // 사용자 번호 시퀀스 값을 가지고 있는 문서에 접근할 수 있는 객체를 가져온다.
                val documentReference = collectionReference.document("UserSequence")
                // 저장할 데이터를 담을 HashMap을 만들어준다.
                val map = mutableMapOf<String, Long>()
                map["value"] = userSequence.toLong()
                // 저장한다.
                documentReference.set(map)
            }
            job1.join()
        }
```

버튼의 아이디로 분기하여 성별값을 반환하는 메서드를 작성해준다.

[AddUserInfoViewModel.kt]

```
    // 성별값을 반환하는 메서드
    fun gettingGender():Gender = when(toggleAddUserInfoGender.value){
        R.id.buttonAddUserInfoMale -> Gender.MALE
        else -> Gender.FEMALE
    }
```

사용자 정보를 저장하는 메서드를 만들어준다.

[UserDao.kt]

```
        // 사용자 정보를 저장한다.
        suspend fun insertUserData(userModel: UserModel){
            val job1 = CoroutineScope(Dispatchers.IO).launch {
                // 컬렉션에 접근할 수 있는 객체를 가져온다.
                val collectionReference = Firebase.firestore.collection("UserData")
                // 컬럭션에 문서를 추가한다.
                // 문서를 추가할 때 객체나 맵을 지정한다.
                // 추가된 문서 내부의 필드는 객체가 가진 프로퍼티의 이름이나 맵에 있는 데이터의 이름과 동일하게 결정된다.
                collectionReference.add(userModel)
            }
            job1.join()
        }
```

사용자 정보를 저장 처리하는 메서드를 구현해준다

[AddUserFragment.kt]

```
    // 데이터를 저장하고 이동한다.
    fun saveUserData(){
        CoroutineScope(Dispatchers.Main).launch {
            // 사용자 번호 시퀀스 값을 가져온다.
            val userSequence = UserDao.getUserSequence()
            // Log.d("test1234", "UserSequence : $userSequence")
            // 시퀀스값을 1 증가시켜 덮어씌워준다.
            UserDao.updateUserSequence(userSequence + 1)

            // 저장할 데이터를 가져온다.
            val userIdx = userSequence + 1
            val userId = arguments?.getString("joinUserId")!!
            val userPw = arguments?.getString("joinUserPw")!!
            val userNickName = addUserInfoViewModel.textFieldAddUserInfoNickName.value!!
            val userAge = addUserInfoViewModel.textFieldAddUserInfoAge.value!!.toInt()
            val userGender = addUserInfoViewModel.gettingGender().num
            val userHobby1 = addUserInfoViewModel.checkBoxAddUserInfoHobby1.value!!
            val userHobby2 = addUserInfoViewModel.checkBoxAddUserInfoHobby2.value!!
            val userHobby3 = addUserInfoViewModel.checkBoxAddUserInfoHobby3.value!!
            val userHobby4 = addUserInfoViewModel.checkBoxAddUserInfoHobby4.value!!
            val userHobby5 = addUserInfoViewModel.checkBoxAddUserInfoHobby5.value!!
            val userHobby6 = addUserInfoViewModel.checkBoxAddUserInfoHobby6.value!!
            val userState = UserState.USER_STATE_NORMAL.num
```

```
            val userState = UserState.USER_STATE_NORMAL.num

            // 저장할 데이터를 객체에 담는다.
            val userModel = UserModel(userIdx, userId, userPw, userNickName, userAge, userGender,
                userHobby1, userHobby2, userHobby3, userHobby4, userHobby5, userHobby6, userState)

            // 사용자 정보를 저장한다.
            UserDao.insertUserData(userModel)

            val materialAlertDialogBuilder = MaterialAlertDialogBuilder(mainActivity)
            materialAlertDialogBuilder.setTitle("가입완료")
            materialAlertDialogBuilder.setMessage("가입이 완료되었습니다\n로그인해주세요")
            materialAlertDialogBuilder.setPositiveButton("확인") { dialogInterface: DialogInterface, i: Int ->
                mainActivity.removeFragment(MainFragmentName.ADD_USER_INFO_FRAGMENT)
                mainActivity.removeFragment(MainFragmentName.JOIN_FRAGMENT)
            }
            materialAlertDialogBuilder.show()
        }
```

메서드를 호출해준다.

```
[AddUserInfoFragment.kt - settingButtonAddUserInfoSubmit]

                    // 모든 유효성 검사에 통과를 했다면..
                    if(chk == true) {
                        // 사용자 정보를 저장한다.
                        saveUserData()

//                          val materialAlertDialogBuilder = MaterialAlertDialogBuilder(mainActivity)
//                          materialAlertDialogBuilder.setTitle("가입완료")
//                          materialAlertDialogBuilder.setMessage("가입이 완료되었습니다\n로그인해주세요")
//                          materialAlertDialogBuilder.setPositiveButton("확인") { dialogInterface: DialogInterface, i: Int ->
//                              mainActivity.removeFragment(MainFragmentName.ADD_USER_INFO_FRAGMENT)
//                              mainActivity.removeFragment(MainFragmentName.JOIN_FRAGMENT)
//                          }
//                          materialAlertDialogBuilder.show()
                    }
```

아이디 중복 확인을 위한 메서드를 만들어준다.

[UserDao.kt]

```
        // 입력한 아이디가 저장되어 있는 문서가 있는지 확인한다(중복처리)
        // 사용할 수 있는 아이디라면 true, 존재하는 아이디라면 false를 반환한다.
        suspend fun checkUserIdExist(joinUserId:String) : Boolean{

            var chk = false

            val job1 = CoroutineScope(Dispatchers.IO).launch {
                // 컬렉션에 접근할 수 있는 객체를 가져온다.
                val collectionReference = Firebase.firestore.collection("UserData")
                // UserId 필드가 사용자가 입력한 아이디와 같은 문서들을 가져온다.
                // whereEqualTo : 같은것
                // whereGreaterThan : 큰것
                // whereGreaterThanOrEqualTo : 크거나 같은 것
                // whereLessThan : 작은 것
                // whereLessThanOrEqualTo : 작거나 같은 것
                // whereNotEqualTo : 다른 것
                // 필드의 이름, 값 형태로 넣어준다
                val queryShapshot = collectionReference.whereEqualTo("userId", joinUserId).get().await()
                // 반환되는 리스트에 담긴 문서 객체가 없다면 존재하는 아이디로 취급한다.
                chk = queryShapshot.isEmpty
            }

            job1.join()

            return chk
        }
```

중복 확인 버튼의 코드를 수정해준다.

```
[JoinFragment.kt - settingButtonJoinCheckId]

                setOnClickListener {

                    CoroutineScope(Dispatchers.Main).launch {
                        checkUserIdExist = UserDao.checkUserIdExist(joinViewModel?.textFieldJoinUserId?.value!!)
                        // Log.d("test1234", "$checkUserIdExist")

                        if(checkUserIdExist == false) {
                            joinViewModel?.textFieldJoinUserId?.value = ""
                            Tools.showErrorDialog(mainActivity, fragmentJoinBinding.textFieldJoinUserId,
                                "아이디 입력 오류", "존재하는 아이디입니다\n다른 아이디를 입력해주세요")
                        } else {
                            val materialAlertDialogBuilder = MaterialAlertDialogBuilder(mainActivity)
                            materialAlertDialogBuilder.setMessage("사용 가능한 아이디 입니다")
                            materialAlertDialogBuilder.setPositiveButton("확인", null)
                            materialAlertDialogBuilder.show()
                        }

                    }
                    // checkUserIdExist = true
                }
```

026 로그인 처리

UserModel 에 매개변수가 없는 생성자를 추가해준다.

```
[UserModel.kt]

    // 매개 변수가 없는 생성자
    // fireStore 를 사용할 때 데이터를 담을 클래스 타입을 지정하게 되면
    // 매개 변수가 없는 생성자를 사용해 객체 생성해주기 때문에 만들어줘야 한다.
    constructor() : this(0, "", "", "", 0, 0, false, false, false, false, false, false, 0)
```

UserDao에 아이디를 받아 정보를 가져오는 메서드를 구현해준다.

[UserDao.kt]

```
        // 아이디를 통해 사용자 정보를 가져오는 메서드
        suspend fun getUserDataById(userId:String) : UserModel?{
            // 사용자 정보 객체를 담을 변수
            var userModel:UserModel? = null

            val job1 = CoroutineScope(Dispatchers.IO).launch {
                // UserData 컬렉션 접근 객체를 가져온다.
                val collectionReference = Firebase.firestore.collection("UserData")
                // userId 필드가 매개변수로 들어오는 userId와 같은 문서들을 가져온다.
                val querySnapshot = collectionReference.whereEqualTo("userId", userId).get().await()
                // 만약 가져온 것이 있다면
                if(querySnapshot.isEmpty == false){
                    // 가져온 문서객체들이 들어 있는 리스트에서 첫 번째 객체를 추출한다.
                    // 아이디가 동일한 사용는 없기 때문에 무조건 하나만 나오기 때문이다
                    userModel = querySnapshot.documents[0].toObject(UserModel::class.java)
                    //Log.d("test1234", "userModel")
                }
            }
            job1.join()

            return userModel
        }
```

로그인 처리 하는 메서드를구현해준다.

[LoginFragment.kt]

```
// 로그인 처리
    fun loginPro(){
        // 사용자가 입력한 정보를 가져온다.
        val userId = loginViewModel.textFieldLoginUserId.value!!
        val userPw = loginViewModel.textFieldLoginUserPw.value!!


        val job1 = CoroutineScope(Dispatchers.Main).launch {
            val loginUserModel = UserDao.getUserDataById(userId)

            // 만약 null 이라면..
            if(loginUserModel == null){
                Tools.showErrorDialog(mainActivity, fragmentLoginBinding.textFieldLoginUserId, "로그인 오류",
                    "존재하지 않는 아이디 입니다")
            }
            // 만약 정보를 가져온 것이 있다면
            else {
                // 입력한 비밀번호와 서버에서 받아온 사용자의 비밀호가 다르다면..
                if(userPw != loginUserModel.userPw){
                    Tools.showErrorDialog(mainActivity, fragmentLoginBinding.textFieldLoginUserPw, "로그인 오류",
                        "비밀번호가 잘못되었습니다")
                }
                // 비밀번호가 일치한다면
                else {
                    Snackbar.make(fragmentLoginBinding.root, "로그인에 성공하였습니다", Snackbar.LENGTH_SHORT).show()

                    // ContentActivity를 실행한다.
                    val contentIntent = Intent(mainActivity, ContentActivity::class.java)
                    startActivity(contentIntent)
                    // MainActivity를 종료한다.
                    mainActivity.finish()
                }
            }
        }
    }
```

호출해준다.

```
[LoginFragment.kt - settingButtonLoginSubmit]

                // 버튼을 눌렀을 때
                setOnClickListener {

                    // 유효성 검사
                    val chk = checkInputForm()

                    // 모든 유효성 검사에 통과를 했다면
                    if(chk == true) {

                        loginPro()

//                          // ContentActivity를 실행한다.
//                          val contentIntent = Intent(mainActivity, ContentActivity::class.java)
//                          startActivity(contentIntent)
//                          // MainActivity를 종료한다.
//                          mainActivity.finish()
                    }
                }
```

로그인 성공 후 ContentActivity를 실행할 때 Intent에 사용자 인덱스와 닉네임을 전달해준다.

```
[LoginFragment.kt - loginPro]

                    // 로그인한 사용자의 정보를 전달해준다.
                    contentIntent.putExtra("loginUserIdx", loginUserModel.userIdx)
                    contentIntent.putExtra("loginUserNickName", loginUserModel.userNickName)
```

ContentActivity에 로그인한 사용자 정보를 담을 변수를 정의한다.

```
[ContentActivity.kt]

    // 로그인한 사용자의 정보를 담을 변수
    var loginUserIdx = 0
    var loginUserNickName = ""
```

로그인한 사용자 정보를 담아둔다.

```
[ContentActivity.kt - onCreate]

        // 로그인한 사용자 정보를 변수에 담아둔다.
        loginUserIdx = intent.getIntExtra("loginUserIdx", 0)
        loginUserNickName = intent.getStringExtra("loginUserNickName")!!
```

자동 로그인 체크시 사용자 정보를 저장하는 코드를 넣어준다.

```
[LoginFragment.kt - loginPro]

                    // 자동 로그인이 체크되어 있다면...
                    if(loginViewModel.checkBoxLoginAuto.value == true){
                        // Preferences 에 사용자 정보를 저장해둔다.
                        val sharedPreferences = mainActivity.getSharedPreferences("AutoLogin", Context.MODE_PRIVATE)
                        val editor = sharedPreferences.edit()
                        editor.putInt("loginUserIdx", loginUserModel.userIdx)
                        editor.putString("loginUserNickName", loginUserModel.userNickName)
                        editor.apply()
                    }
```

MainActivity에 자동 로그인 관련 코드를 넣어준다.

```
[MainActivity.kt - onCreate]

        // 자동로그인시 저장된 사용자 정보를 가져온다.
        val sharedPreferences = getSharedPreferences("AutoLogin", MODE_PRIVATE)
        val loginUserIdx = sharedPreferences.getInt("loginUserIdx", -1)
        val loginUserNickName = sharedPreferences.getString("loginUserNickName", null)

        // 자동 로그인시 저장된 사용자 인덱스값이 없다면(자동로그인을 체크하지 않았다면)
        if(loginUserIdx == -1) {
            // 첫 화면을 띄워준다.
            replaceFragment(MainFragmentName.LOGIN_FRAGMENT, false, false, null)
        }
        // 그렇지 않으면
        else {
            // ContentActivity를 실행한다.
            val contentIntent = Intent(this, ContentActivity::class.java)

            // 로그인한 사용자의 정보를 전달해준다.
            contentIntent.putExtra("loginUserIdx", loginUserIdx)
            contentIntent.putExtra("loginUserNickName", loginUserNickName)

            startActivity(contentIntent)
            // MainActivity를 종료한다.
            finish()
        }
```

027 글 작성 처리

상단에 이미지 첨부 여부를 담을 변수를 정의한다.

```
[AddContentFragment.kt]

    // 이미지를 첨부한 적이 있는지..
    var isAddPicture = false
```

입력 요소 설정 메서드에 변수의 값을 **false**로 설정한다.

```
[AddContentFragment.kt - settingInputForm]

        isAddPicture = false
```

카메라로 사진을 촬영해 이미지를 셋팅하는 곳에 변수의 값을 **true**로 설정한다.

```
[AddContentFragment.kt - settingCarmeraLauncher]

                isAddPicture = true
```

앨범을 통해 사진을 촬영해 이미지를 셋팅하는 곳에 변수의 값을 **true**로 설정한다.

```
[AddContentFragment.kt - settingAlbumAauncher]

isAddPicture = true
```

이미지 뷰의 이미지를 저장하는 메서드를 만든다.

[Tools.kt]

```
        // 이미지뷰의 이미지를 추출해 로컬에 저장한다.
        fun saveImageViewData(context:Context, imageView:ImageView, fileName:String){
            // 외부 저장소까지의 경로를 가져온다.
            val filePath = context.getExternalFilesDir(null).toString()
            // 이미지 뷰에서 BitmapDrawable 객체를 추출한다.
            val bitmapDrawable = imageView.drawable as BitmapDrawable

            // 로컬에 저장할 경로
            val file = File("${filePath}/${fileName}")
            // 스트림 추출
            val fileOutputStream = FileOutputStream(file)
            // 이미지를 저장한다.
            // 첫 번째 : 이미지 데이터 포멧(JPEG, PNG, WEBP_LOSSLESS, WEBP_LOSSY)
            // 두 번째 : 이미지의 퀄리티
            // 세 번째 : 이미지 데이터를 저장할 파일과 연결된 스트림
            bitmapDrawable.bitmap.compress(Bitmap.CompressFormat.JPEG, 100, fileOutputStream)
            fileOutputStream.flush()
            fileOutputStream.close()
        }
```

ContentDao.kt 파일을 만들어준다.

이미지를 업로드 하는 메서드를 만들어준다.

[ContentDao.kt]

```
        // 이미지 데이터를 firebase storage에 업로드는 메서드
        suspend fun uploadImage(context:Context, fileName:String, uploadFileName:String){
            // 외부저장소 까지의 경로를 가져온다.
            val filePath = context.getExternalFilesDir(null).toString()
            // 서버로 업로드할 파일의 경로
            val file = File("${filePath}/${fileName}")
            val uri = Uri.fromFile(file)

            val job1 = CoroutineScope(Dispatchers.IO).launch {
                // Storage에 접근할 수 있는 객체를 가져온다.(폴더의 이름과 파일이름을 저장해준다.
                val storageRef = Firebase.storage.reference.child("image/$uploadFileName")
                // 업로드한다.
                storageRef.putFile(uri)
            }

            job1.join()
        }
```

글 작성 처리 메서드에 이미지 업로드 코드를 작성해준다.

[AddContentFragment.kt]

```
    // 글 작성처리 메서드
    fun uploadContentData(){
        CoroutineScope(Dispatchers.Main).launch {
            // 첨부된 이미지가 있다면
            if(isAddPicture == true) {
                // 이미지의 뷰의 이미지 데이터를 파일로 저장한다.
                Tools.saveImageViewData(contentActivity, fragmentAddContentBinding.imageViewAddContent, "uploadTemp.jpg")
                // 서버에서의 파일 이름
                val serverFileName = "image_${System.currentTimeMillis()}.jpg"
                // 서버로 업로드한다.
                ContentDao.uploadImage(contentActivity, "uploadTemp.jpg", serverFileName)
            }
        }
    }
```

완료 메뉴를 눌렀을 때 글 데이터를 업로드 하도록 한다.

```
[AddContentFragment.kt - settingToolbarAddContent]

                                // 글 데이터를 업로드한다.
                                uploadContentData()
                                // ReadContentFragment로 이동한다.
                                // contentActivity.replaceFragment(ContentFragmentName.READ_CONTENT_FRAGMENT, true, true,
null)
```

게시글의 상태를 나타내는 값을 정의한다.

```
[Tools.kt]

// 게시글 종류를 나타내는 값을 정의한다.
enum class ContentState(var str:String, var number:Int){
    CONTENT_STATE_NORMAL("정상", 1),
    COTTENT_STATE_REMOVE("삭제", 2),
}
```

게시글 정보를 담을 Model을 작성해준다.

```
[ContentModel.kt]

package kr.co.lion.androidproject4boardapp.model

data class ContentModel(var contentIdx:Int, var contentSubject:String, var contentType:Int,
    var contentText:String, var contentImage:String?, var contentWriterIdx:Int, var contentWriteDate:String,
    var contentState:Int){

    constructor():this(0, "", 0, "", "", 0, "", 0)
}
```

게시글 시퀀스 값을 가져오는 메서드를 만들어준다.

[ContentDao.kt]

```
        // 게시글 번호 시퀀스값을 가져온다.
        suspend fun getContentSequence():Int{

            var contentSequence = -1

            val job1 = CoroutineScope(Dispatchers.IO).launch {
                // 컬렉션에 접근할 수 있는 객체를 가져온다.
                val collectionReference = Firebase.firestore.collection("Sequence")
                // 게시글 번호 시퀀스 값을 가지고 있는 문서에 접근할 수 있는 객체를 가져온다.
                val documentReference = collectionReference.document("ContentSequence")
                // 문서내에 있는 데이터를 가져올 수 있는 객체를 가져온다.
                val documentSnapShot = documentReference.get().await()
                contentSequence = documentSnapShot.getLong("value")?.toInt()!!
            }
            job1.join()

            return contentSequence
        }
```

게시글 시퀀스 값을 변경하는 메서드를 만들어준다.

[ContentDao.kt]

```
        // 게시글 시퀀스 값을 업데이트 한다.
        suspend fun updateContentSequence(contentSequence:Int){
            val job1 = CoroutineScope(Dispatchers.IO).launch {
                // 컬렉션에 접근할 수 있는 객체를 가져온다.
                val collectionReference = Firebase.firestore.collection("Sequence")
                // 게시글 번호 시퀀스 값을 가지고 있는 문서에 접근할 수 있는 객체를 가져온다.
                val documentReference = collectionReference.document("ContentSequence")
                // 저장할 데이터를 담을 HashMap을 만들어준다.
                val map = mutableMapOf<String, Long>()
                map["value"] = contentSequence.toLong()
                // 저장한다.
                documentReference.set(map)
            }
            job1.join()
        }
```

게시판 타입 버튼의 아이디에 따라 타입값을 반환하는 메서드를 만들어준다

[AddContentViewModel.kt]

```
    // MutableLiveData에 담긴 버튼의 ID 값을 통해 게시판 타입값을 반환한다.
    fun gettingContentType():ContentType = when(toggleAddContentType.value){
        R.id.buttonAddContentType1 -> ContentType.TYPE_FREE
        R.id.buttonAddContentType2 -> ContentType.TYPE_HUMOR
        R.id.buttonAddContentType3 -> ContentType.TYPE_SOCIETY
        R.id.buttonAddContentType4 -> ContentType.TYPE_SPORTS
        else -> ContentType.TYPE_ALL
    }
```

글 작성 처리 메서드에 글을 올리는 코드를 구현해준다.

[AddContentFragment.kt]

```
    // 글 작성처리 메서드
    fun uploadContentData(){
        CoroutineScope(Dispatchers.Main).launch {

            // 서버에서의 첨부 이미지 파일 이름
            var serverFileName:String? = null

            // 첨부된 이미지가 있다면
            if(isAddPicture == true) {
                // 이미지의 뷰의 이미지 데이터를 파일로 저장한다.
                Tools.saveImageViewData(contentActivity, fragmentAddContentBinding.imageViewAddContent, "uploadTemp.jpg")
                // 서버에서의 파일 이름
                serverFileName = "image_${System.currentTimeMillis()}.jpg"
                // 서버로 업로드한다.
                ContentDao.uploadImage(contentActivity, "uploadTemp.jpg", serverFileName)
            }

            // 게시글 시퀀스 값을 가져온다.
            val contentSequence = ContentDao.getContentSequence()
            // 게시글 시퀀스 값을 업데이트 한다.
            ContentDao.updateContentSequence(contentSequence + 1)

            // 업로드할 정보를 담아준다.
            val contentIdx = contentSequence + 1
            val contentSubject = addContentViewModel.textFieldAddContentSubject.value!!
            val contentType = addContentViewModel.gettingContentType().number
            val contentText = addContentViewModel.textFieldAddContentText.value!!
            val contentImage = serverFileName
```

```
            val contentImage = serverFileName
            val contentWriterIdx = contentActivity.loginUserIdx

            val simpleDateFormat = SimpleDateFormat("yyyy-MM-dd")
            val contentWriteDate = simpleDateFormat.format(Date())
            val contentState = ContentState.CONTENT_STATE_NORMAL.number

            val contentModel = ContentModel(contentIdx, contentSubject, contentType, contentText, contentImage,
contentWriterIdx, contentWriteDate, contentState)
            // 업로드한다.
            ContentDao.insertContentData(contentModel)

            // ReadContentFragment로 이동한다.
            Tools.hideSoftInput(contentActivity)
            contentActivity.replaceFragment(ContentFragmentName.READ_CONTENT_FRAGMENT, true, true, null)
        }
    }
```