Null Safety

null로 인해 프로그램 전체 혹은 앱 전체가 멈출 수 있다. => 안전장치 : Null Safety

null을 입력하기 위해서는 var 변수명: **타입?**

① 함수 파라미터로 null 사용시 해당 파라미터에 대해서 null체크 필요

```
fun 함수명(str:String?){
    if(str != null){
    var length2 = str.length
    }
}
```

② 함수의 리턴타입에도 null 허용여부 설정

```
fun 함수명(): String?{
    if(str != null){
    var length2 = str.length
    }
}
```

안전한 호출(?.): Nuallable(널 사용가능한)을 위해 null 체크 간결하게 하기

```
fun 함수명(str:String?){
    var result:Int = str?.length?:0
    //str이 null이아니면 str.length을 가져오고 str이 null이면 다음 프로퍼티,
    메소드를 호출하지 않고 그 즉시 null반환
    //str.length가 null일 경우 ?: 오른쪽값인 0이 반환됩니다
}
```

지연초기화

① 변수만 nullable로 미리 선언하고 초기화 나중에 할 경우 **lateinit** 키워드

`lateinit var 변수: 타입 후 나중에 값 할당(값 할당 전에 변수사용시 null예외 발생)`

② val을 사용하는 지연초기화, 선언 후 최초 사용하는 시점에 초기화 할 경우 **by lazy{초기화 값}**

(초기화하는데 사용하는 리소스 크면 전체 처리속도에 영향, 초기화할 값이 단순할수록 by lazy를 사용하기 좋음)

`val 변수: 타입 by lazy{초기화할 값}`

ACTIVITIY

컨텍스트: 시스템을 이용하기 위한 프로퍼티&메소드가 담겨있는 클래스이자 기본기능이 담김

① Application Context: 애플리케이션과 관련된 핵심기능 담고 있는 클래스, 한 개만 생성됨

② Base Context: 액티비티,서비스, 컨텐트 프로바이더, 브로드캐스트리시버의 기반 클래스
컴포넌트 개수만큼 Base 컨텍스트도 함께 생성 = 호출시점에 따라 다른 컨텍스트 호출

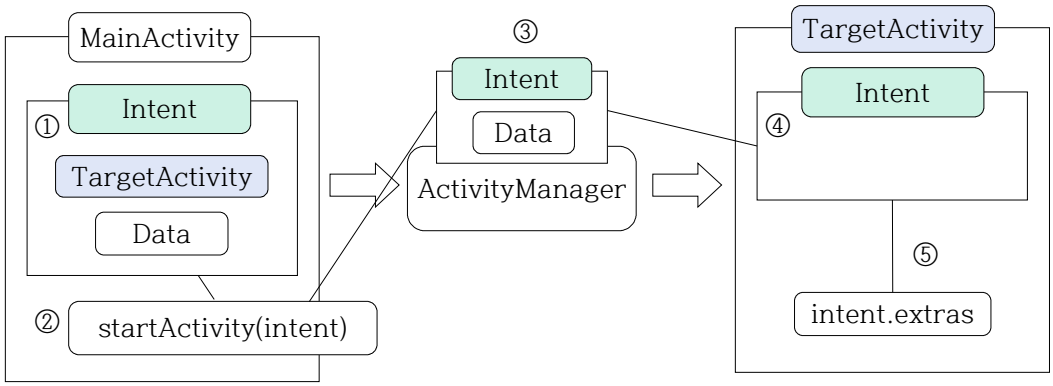
※ Application 클래스

애플리케이션 전반에 걸쳐 접근할 수 있는 전역 상태를 저장하고 관리하는 데 사용
애플리케이션의 생명주기 메서드(예: onCreate, onTerminate, onLowMemory)를 오버라이드하여 앱의 시작, 종료 및 메모리 부족 상태 등을 관리할 수 있습니다.

activity는 context을 상속받아 만들어지며고 사용자가 보고 입력하는 화면을 뜻함

새 액티비티 생성하기
① [app]-[MainActivity.kt 파일이 들어있는 패키지명/]-[우클릭]-[Activity]-[Empty Activity]
② 생성창에서 Activity파일명 입력(xml파일인 레이아웃 파일은 입력한 Activity명에 자동으로 채워짐)

Intent: 액티비티 실행시 필요(MainActivity는 자동등록)



- MainActivity에서 TargetActivity 실행

MainActivity의 onCreate(){}안에 뷰바인딩 후 작성

```
val intent = Intent(this,TargetActivity::class.java) //intent 생성
startActivity(intent) //intent를 넘겨줌
```

- 서로 값 주고받기

MainActivity의 onCreate(){}안에 뷰바인딩 후 작성

```
val intent = Intent(this,TargetActivity::class.java) //intent 생성
intent.putExtra("키", "값") //인텐트에 값 담기
startActivity(intent) //intent를 넘겨줌
```

TargetActivity의 onCreate(){}안에 뷰바인딩 후 작성

```
intent.getStringExtra(key) //인텐트에 담은 키 꺼내기 (해당되는 값이 문자열임)
```

- TargetActivity가 종료되면 MainActivity에 값 돌려주기

① TargetActivity의 onCreate(){

```
val returnIntent = Intent() //인텐트 생성
returnIntent.putExtra(key,value) // 값 저장하기
setResult(RESULT_OK,returnIntent) //상태값과 인텐트 전달, 처리한 결과값에 따라 다름
finish() //TargetActivity 종료
}
```

② build.gradle(Module:app)에서

```
dependencies{
    def depency_version = "1.3.1"
        implementation "androidx.activity:activity-ktx:$dependency_version"
        implementation "androidx.fragment:fragment-ktx:$dependency_version"
}
```

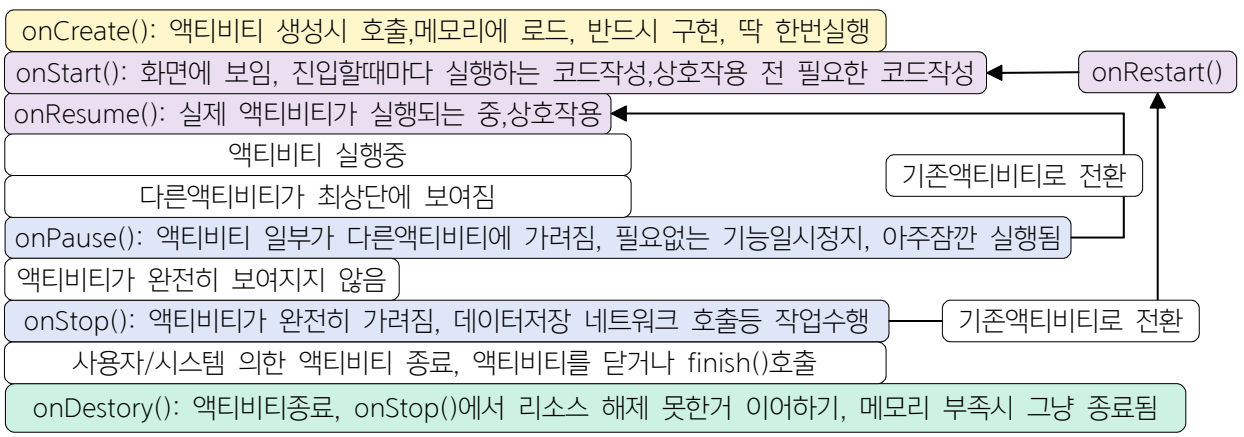
③ MainActivity에서

```
val activityResult = registerForActivityResult(ActivityResultContracts.StartActivity
ForResult()){
    if(it.resultCode == RESULT_OK){
        //정상이면 코드실행
    }
}
```

- launch() 메서드 사용하기 : startActivity()sms 값을 돌려받을 수 없다.

MainActivity가 onCreate(){}안에서 startActivity()를 activityResult.launch(intent)로 바꾸기

ACTIVITIY Lifecycle



ACTIVITIY BackStack

백스택: 액티비티, 화면 컴포넌트를 담는 안드로이드 저장공간, 가장 위에 있는 액티비티를 보게 됨
- 뒤로가기, 현 액티비티 종료시 스택에서 제거됨

Fragment

화면을 분할해서 독립적인 코드로 구성할 수 있게 함, 액티비티의 일부

① FrameLayout 사용(화면전환 o시)

[NEW]-[Fragment]-[Fragment(blank)]-[Fragment 이름 작성]

```
class fragment:Fragement(){
    lateinint var fragmentbinding:FragementBinding
    overrid fun onCreate(inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?): View? {
        return fragmentbinding.root
    }
}
```

MainActivity-activity_main.xml에서 <FrameLayout>으로 넣어주기

MainActivity의 oncreate()에 setFragment() 써놓기 //함수명은 마음대로
oncreate() 바깥에 fun setFragment(){코드작성}하기

```
fun setFragment() {
    val listFragment: listFragment = ListFragment()
    val transaction = supportFragmentManager.beginTransaction()
    transaction.add(R.id.FrameLayout,listFragment)
    transaction.commit()

}
```

① fragmentContainerView 사용

[NEW]-[Fragment]-[Fragment(blank)]-[Fragment 이름 작성]

```
class fragment:Fragement(){
    lateinint var fragmentbinding:FragementBinding
    overrid fun onCreate(inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?): View? {
        return fragmentbinding.root
    }
}
```

MainActivity-activity_main.xml에서 <FrameLayout>을 <fragment>로

MainActivity의 oncreate()에 setFragment() 써놓기 //함수명은 마음대로
oncreate() 바깥에 fun setFragment(){코드작성}하기

```
fun setFragment() {
    val listFragment: listFragment = ListFragment()
    val transaction = supportFragmentManager.beginTransaction()
    transaction.add(R.id.FrameLayout,listFragment)
    transaction.commit()

}
```

File I/O

안드로이드는 리눅스 위에 가상머신이 동작(= 내부적으로 리눅스 기반 파일시스템으로 구성)

① 내부저장소: 특정 앱의 사용자가 접근할 수 있는 영역, 설치한 앱에 제공되는 디렉터리
앱에서 사용하는 데이터를 저장하는 용도
② 외부저장소:ahems 앱이 함께 사용할 수 있는 공간, 접근시 권한을 명세해야함,앱과는 별개

• val file = File("경로")

내부저장소 files 디렉토리에서 읽고 쓰기할경우
• val file = File(filesDir,"파일명") -> 프래그먼트에서 사용시 File(baseContext.filesDir,"파일명")

file.exists(): File 존재여부
file.isFile: 전달된 경로가 파일경로인지?
file.isDirectory: 전달된 경로가 디렉토리경로인지?
file.name: 파일 혹은 디렉토리 이름 반환?
file.createNewFile(): 해당 경로에 파일이 존재하지 않으면 파일 생성
file.mkdir(): 디렉토리 생성
file.delete(): 파일, 빈디렉토리를 제거
file.absolutePath: 파일, 디렉토리 절대경로

파일 읽기

[app]-[java]-우클릭-[New]-[kotlin File/class]-[이름에 fileUtil 입력]

```
fun readTextFile(fullPath: String):String{
    val file = File(fullPath)
    If(!file.exists()) return ""

    val reader =FileReader(file)
    val buffer = BufferedReader(reader)

    var tmp =""
    val result = StringBuffer()

    while(true){
        tmp = buffer.readLine()
        if (tmp == null) break;
        else result.append(buffer)
    }
    buffer.close()
```

```
        return result.toString()
    }
```

파일 쓰기

```
    fun writeTextFile(directory: String,filename: String,content: String):String{
        val dir = File(directory)
        If(!dir.exists()) dir.mkdirs()

        val writer =FileWriter(directory+"/"+filename)
        val buffer = BufferedWriter(reader)
        buffer.write(content)
        buffer.close()

    }
```

SharedPreferences

간단한 데이터 저장 목적으로 제공, 내부저장소를 이용하기 때문에 권한 필요 없음

데이터저장하기

① sharedPreference 생성

```
val shared = getSharedPreferenxces("저장될 파일명". Context.MODE_PRIVATE)
```

② Editor 꺼내기

```
val editor = shared.edit()
```

③ putXxx()로 저장하기

```
editor.putXxx(key,value)
```

④ apply(): 파일에 비동기적으로 저장함(비슷한 의미의 commit()도 있지만 얘는 짧은시간동안 화면이 멈출수 있어서 apply()을 사용)

```
editor.apply()
```

데이터저장하기

① sharedPreference 생성

```
val shared = getSharedPreferenxces("이름". Context.MODE_PRIVATE)
```

② putXxx()로 꺼내기

```
shared.getString(key, 기본값)
```

SharedPreferences

fileListActivity.kt

```
package com.android.memo_file

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.os.Environment
import android.util.Log
import android.widget.ArrayAdapter
import com.android.memo_file.databinding.ActivityFilelistBinding
import java.io.File

class FilelistActivity : AppCompatActivity() {


    lateinit var binding: ActivityFilelistBinding
    lateinit var rootSD: String
    lateinit var fileDir: File
    lateinit var fileList: ArrayList<String>

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

        Log.d("myCheck", "FilelistActivity의onCreate 메서드 실행")


        binding = ActivityFilelistBinding.inflate(layoutInflater)
        setContentView(binding.root)

        rootSD = Environment.getExternalStoragePublicDirectory
        (Environment.DIRECTORY_DOCUMENTS).toString()

        fileDir = File(rootSD)
        fileList = ArrayList<String>()

        var list = fileDir.listFiles()

        for(i in 0 until list.size) {
            if(list[i].name != "profileInstalled" || !list[i].isDirectory )  {
            fileList.add(list[i].name.toString())
            }
        }


        binding.listView.adapter = ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,
        fileList )

        binding.listView.setOnItemClickListener() { adapterView, view, i, l ->

        var returnIntent = Intent()
        returnIntent.putExtra("fileNameToOpen", fileList[i])
        setResult(RESULT_OK, returnIntent)

        finish()
        }

    }
}
```

FilenameActivity

```
package com.android.memo_file

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import com.android.memo_file.databinding.ActivityFilenameBinding

class FilenameActivity : AppCompatActivity() {

    lateinit var binding: ActivityFilenameBinding

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)

        binding = ActivityFilenameBinding.inflate(layoutInflater)
        setContentView(binding.root)


        MyApplication.preferences.setString("MyKey", "FilenameActivity에서 값을 변경하다. ")
        val temp = MyApplication.preferences.getString("MyKey", "")
        Log.d("myCheck", "SharedPreference에서MyKey의 값은 ${temp}")

        binding.button4.setOnClickListener() {

            val returnIntent = Intent()
            returnIntent.putExtra("fileNameToSave", binding.editText2.text.toString())
            setResult(RESULT_OK, returnIntent)
            finish()

        }
    }
}
```