State Management , App Architect MVVM

1 Part 1

1.1 exercise 1 [ViewModel Configuration]

install dependancy of View Model inside gradle module app create a class and extend it to use view model characteristics

Code Input Section

```

    implementation("androidx.lifecycle:lifecycle-runtime-ktx:2.7.0")
    implementation("androidx.lifecycle:lifecycle-viewmodel-compose:2.7.0")
```

1.2 exercise 1 [Defining student class]

generate a class of student to represent the student data on real life

Code Input Section

```


data class Student(val name: String, val age: Int, val speciality: String)
```

1.3 exercise 1 [making of Dummy Database]

in order to handle student data , we will use the dummy database that generate a list of students on application startup

Code Input Section

```

class DummyDatabase() {


    fun getStudent(): Student {
        return Student("ibrahim", 24, "computer science")
    }


    fun populateDatabase(count: Int): MutableList<Student> {
        val names = arrayOf("amine", "nihal", "khadidja", "ahmed")
        val ages = arrayOf(24, 22, 23, 26)
        val specialities =
            arrayOf("computer scinece", "Science technology", "medecine", "paramedic")

        val students = mutableListOf<Student>()
        for (i in 0..count) {
            val randomNumber = Random.nextInt(0, 4)
            students.add(
                Student(
                    names[randomNumber],
                    ages[randomNumber],
                    specialities[randomNumber]
                )
            )
        }
        return students
    }

}
```

1.4 Student app view model

the student view model hold student state and expose the action that this app should have on the students

```

class StudentListViewModel() : ViewModel() {
    val students = mutableStateListOf<Student>()
    val db = DummyDatabase()

    init {
        students.addAll(db.populateDatabase(/*TODO: number of students */))
    }

    fun addStudent(){
        students.add(db.getStudent())
    }

    fun removeStudent(index:Int){
        students.removeAt(index)
    }



}
```

2 Part 2: UI Scripts

2.1 speciality entry Script

```


@Composable
fun SpecialityEntry(specialityName: String ="speciality) {

    Card(modifier = Modifier.width(100.dp)) {
        Column(
            Modifier.padding(10.dp),
            verticalArrangement = Arrangement.spacedBy(5.dp)
        ) {
            //icon
            Icon(imageVector = Icons.Default.AccountBox, contentDescription = null)

            // category name
            Row(horizontalArrangement = Arrangement.spacedBy(10.dp)) {
                Text(text = specialityName, style = MaterialTheme.typography.bodyLarge)
                Text(text = "(03)")
            }
        }
    }

}
```

2.2 Student entry script

```

@Composable
fun StudentEntry(student: Student, removeStudentAction: () -> Unit) {
    Card(Modifier.fillMaxWidth()) {
        Row(
            Modifier
                .padding(10.dp)
                .fillMaxWidth(),
            horizontalArrangement = Arrangement.SpaceAround,
            verticalAlignment = Alignment.CenterVertically

        ) {
            Icon(
```

```
                imageVector = Icons.Default.Person, contentDescription = null, Modifier.weight(.5f)
            )
            Column(verticalArrangement = Arrangement.SpaceAround, modifier = Modifier.weight(2f)) {
                Text(text = student.name)
                Text(text = student.speciality)
            }
            Row(
                Modifier.weight(.5f),
                horizontalArrangement = Arrangement.spacedBy(5.dp),
                verticalAlignment = Alignment.CenterVertically
            ) {
                Text(text = student.age.toString())
                IconButton(onClick = { removeStudentAction.invoke() }) {
                    Icon(imageVector = Icons.Default.Delete, contentDescription = null)
                }
            }

        }
    }
}
```

2.3 Student List Screen

```
  @Preview
@Composable
fun StudentListScreen() {

    val student_vm = viewModel<StudentListViewModel>()
    val students = remember {
        student_vm.students
    }


    val specialities = listOf("MI", "ST", "SNV")

    Surface {
        Scaffold(
            topBar = {
                TopAppBar(title = { Text(text = "Student list") },
                    navigationIcon = {
                        IconButton(onClick = { student_vm.addStudent() }) {
                            Icon(imageVector = Icons.Default.Menu, contentDescription = null)
                        }
                    },
                    actions = {
                        IconButton(onClick = { /*TODO*/ }) {
                            Icon(imageVector = Icons.Default.Delete, contentDescription = null)
                        }
                        IconButton(onClick = { /*TODO*/ }) {
                            Icon(imageVector = Icons.Default.Edit, contentDescription = null)
                        }
                        IconButton(onClick = { /*TODO*/ }) {
                            Icon(imageVector = Icons.Default.Share, contentDescription = null)
                        }
                    })
            },
            floatingActionButtonPosition = FabPosition.Center,
            floatingActionButton = {
                FloatingActionButton(
                    onClick = { student_vm.addStudent() },
                    Modifier.padding(10.dp)
                ) {
```

```
                    Row(Modifier.padding(5.dp)) {
                        Icon(imageVector = Icons.Default.Add, contentDescription = null)
                        Text(text = "Add Student")
                    }
                }
            }
        ) {
            Column(Modifier.padding(it), verticalArrangement = Arrangement.spacedBy(10.dp)) {
                LazyRow(
                    horizontalArrangement = Arrangement.SpaceAround,
                    modifier = Modifier.fillMaxWidth()
                ) {
                    items(specialities) { specialityName ->
                        SpecialityEntry(specialityName)
                    }
                }
                if (students.isNullOrEmpty()) Box(modifier = Modifier.fillMaxWidth()) {
                    Text(text = "No student is added yet", Modifier.align(Alignment.Center))
                } else {
                    Text(text = "Student List", style = MaterialTheme.typography.titleMedium)
                    LazyColumn(
                        verticalArrangement = Arrangement.spacedBy(10.dp),
                        modifier = Modifier.padding(horizontal = 10.dp)
                    ) {
                        itemsIndexed(students) { index, student ->
                            StudentEntry(student = student) { student_vm.removeStudent(index) }
                        }
                    }

                }

            }

        }

    }

}
```