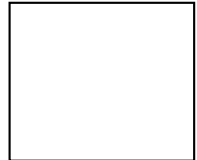




**LABORATORIUM PEMBELAJARAN ILMU KOMPUTER**  
**FAKULTAS ILMU KOMPUTER**  
**UNIVERSITAS BRAWIJAYA**

BAB : PENGANTAR PEMROGRAMAN REAKTIF  
NAMA : PUTUT GURITNO BRAMANTYO  
NIM : 145150207111061  
KELOMPOK : 4  
TANGGAL : 25/11/2020  
ASISTEN : - IMAN HARIE NAWANTO  
- ERSYA NADIA CANDRA



## PROSEDUR 1

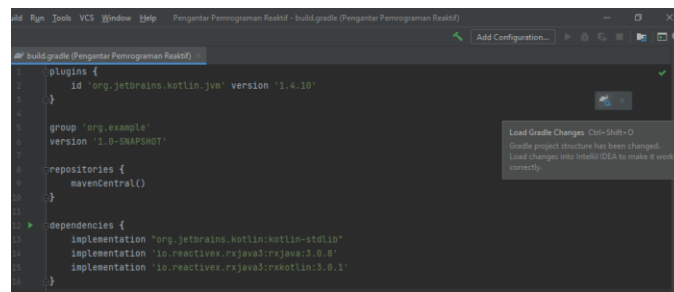
### A. Soal

Memasukkan pustaka RxJava dan RxKotlin pada proyek IntelliJ

### B. Source Code

```
dependencies {  
    implementation "org.jetbrains.kotlin:kotlin-stdlib"  
    implementation "io.reactivex.rxjava3:rxjava:3.0.8"  
    implementation "io.reactivex.rxjava3:rxcotlin:3.0.1"  
}
```

### C. Screenshot



### D. Penjelasan

Langkah awal yaitu masuk ke Maven Central kemudian di search memasukkan kata kunci: rxjava dan pilih : io.reactivex.rxjava3 kemudian pilih versi terkini rxjava pada kolom Latest Version. Setelah itu copy teks pada Graddle Groovy DSL dan tambahkan ke dependence

Kemudian kembali ke Maven Central dan search : rxkotlin lalu ulangi seperti langkah rxjava

Setelah selesai, klik tombol “Load Gradle Changes” yang ada di kanan atas dan tunggu hingga proses download selesai

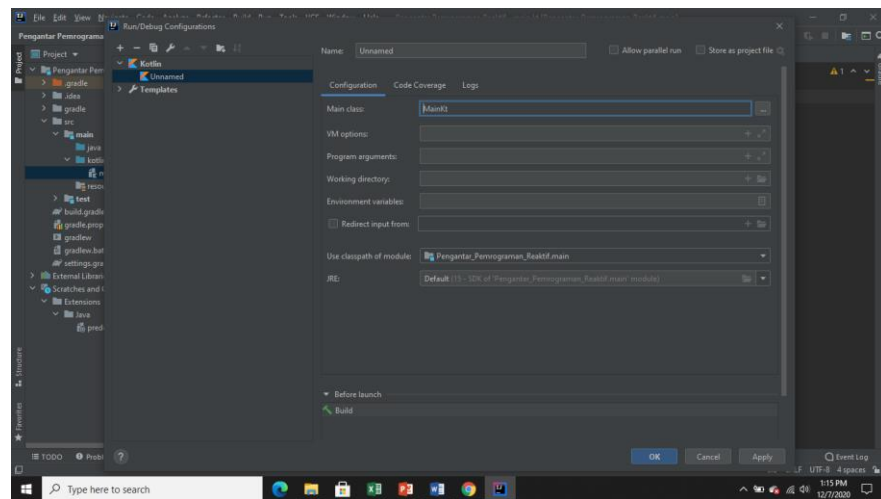
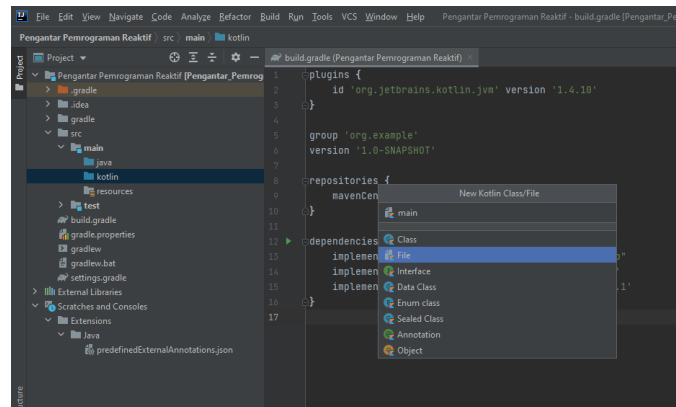
## PROSEDUR 2.1

### E. Soal

Konfigurasi Run

### F. Source Code


## G. Screenshot



## H. Penjelasan

Langkah pertama adalah membuat file main.kt pada src/main/kotlin. Setelah selesai membuat main.kt dan menulis program, klik menu Run lalu pilih Edit Configurations. Pada dialog yang muncul, klik tanda + dan pilih Kotlin. Pada baris Use classpath of module pilih Pengantar\_Pemrograman\_Reaktif.main dan pada baris Main class pilih MainKt lalu OK

## PROSEDUR 2.2

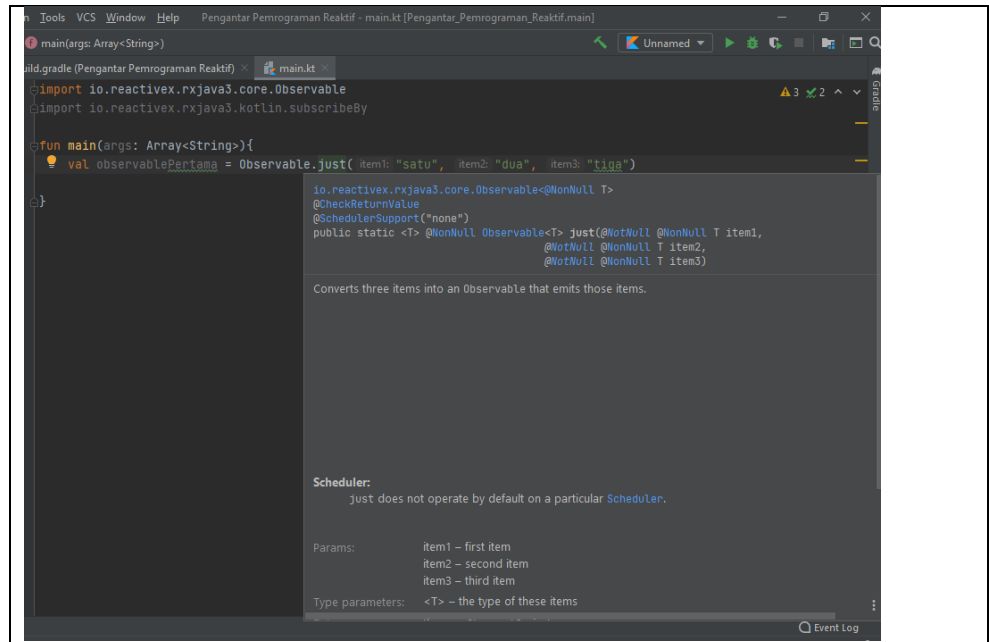
1. Pada file main.kt, tambahkan kode sbb:

```
import io.reactivex.rxjava3.core.Observable

fun main(args: Array<String>){
    val observablePertama = Observable.just("satu", "dua", "tiga")
}
```

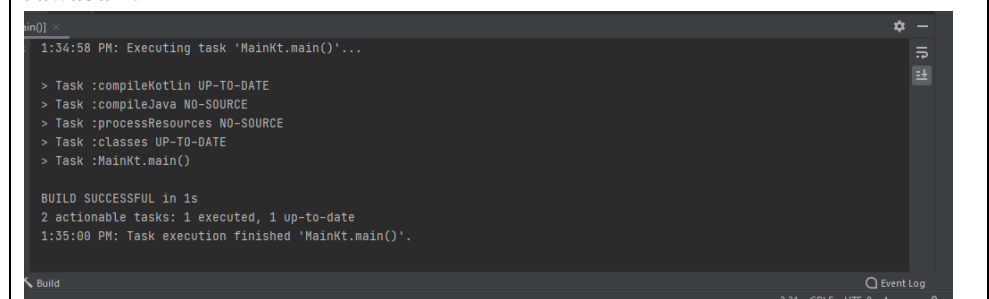
2. Letakkan typing cursor (yang berkedip) di dalam kata just, kemudian letakkan kursor mouse diatasnya. Ambil tangkapan layar dan tempel/paste di kolom jawaban sbb:

Jawaban :



3. Jalankan kode tersebut dengan menekan tombol Run, apa yang tampil pada IDE?

Jawaban :



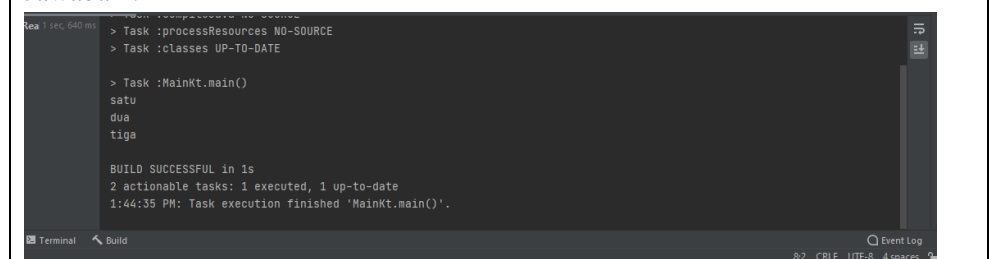
4. Tambahkan kode menjadi sbb:

```
import io.reactivex.rxjava3.core.Observable

fun main(args: Array<String>){
    val observablePertama = Observable.just("satu", "dua", "tiga")
    val subscriberPertama = observablePertama.subscribe{
        println(it)
    }
}
```

5. Jalankan kode tersebut dengan menekan tombol Run, apa yang tampil pada IDE?

Jawaban :



6. Apakah kesimpulan yang dapat saudara tarik?

Jawaban :

Pada observable pertama, variable dideklarasikan terlebih dahulu. Ketika di run tidak akan ada hasil karena tidak ada fungsi untuk menampilkan output. Pada subscriber pertama, observable pertama dipanggil kembali dan ditampilkan dengan perintah `println(it)`

7. Tambahkan kode sbb:

```
import io.reactivex.rxjava3.core.Observable

fun main(args: Array<String>){
    val observablePertama = Observable.just("satu", "dua", "tiga")
    val subscriberPertama = observablePertama.subscribe{
        println(it)
    }
    val subscriberKedua = observablePertama.subscribe{
        println(it)
    }
}
```

8. Jalankan kode tersebut dengan menekan tombol Run, apa yang tampil pada IDE?

Jawaban :

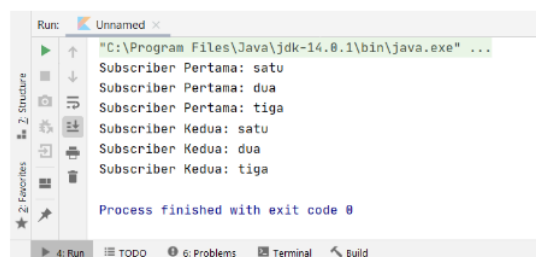


9. Apakah kesimpulan yang dapat saudara tarik?

Jawaban :

Sama seperti subscriber pertama, subscriber kedua digunakan untuk memanggil variable pada observable pertama

10. Ubahlah kode di atas, sehingga output berubah menjadi:



Jawaban :

The screenshot shows an IDE window with a Kotlin file named `main.kt`. The code defines an `Observable` with three items: "satu", "dua", and "tiga". It then creates two subscribers, `subscriberPertama` and `subscriberKedua`, both of which subscribe to the `observablePertama` and print their respective values. The IDE's output window shows the successful compilation and execution of the code, displaying the output of the subscribers: "Subscriber Pertama : satu", "Subscriber Pertama : dua", "Subscriber Pertama : tiga", "Subscriber Kedua : satu", "Subscriber Kedua : dua", and "Subscriber Kedua : tiga".

```
import io.reactivex.rxjava3.core.Observable

fun main(args: Array<String>){
    val observablePertama = Observable.just(item1: "satu", item2: "dua", item3: "tiga")
    val subscriberPertama = observablePertama.subscribe{ it: String!
        println("Subscriber Pertama : " + it)
    }
    val subscriberKedua = observablePertama.subscribe{ it: String!
        println("Subscriber Kedua : " + it)
    }
}
```

Task :compileJava NO-SOURCE  
Task :processResources NO-SOURCE  
Task :classes UP-TO-DATE  
Task :MainKt.main()  
Subscriber Pertama : satu  
Subscriber Pertama : dua  
Subscriber Pertama : tiga  
Subscriber Kedua : satu  
Subscriber Kedua : dua  
Subscriber Kedua : tiga  
BUILD SUCCESSFUL in 814ms  
2 actionable tasks: 1 executed, 1 up-to-date  
1:55:36 PM: Task execution finished 'MainKt.main()'.

11. Sisipkan kode berikut ini setelah variabel `subscriberKedua`:

```
val subscriberKetiga = observablePertama.subscribeBy(
    onNext = { println(it)},
    onComplete = { println("selesai")},
    onError = {println("gangguan")}
)
```

12. Dari library/package apakah method `subscribeBy` berasal?

Jawaban : Function

13. Jalankan kode tersebut dengan menekan tombol Run, apa yang tampil pada IDE?

Jawaban :

The screenshot shows the IDE's output window after running the code. It displays the same output as the first screenshot, but with an additional "selesai" message at the end, indicating the completion of the observable sequence. The output is: "Subscriber Pertama : satu", "Subscriber Pertama : dua", "Subscriber Pertama : tiga", "Subscriber Kedua : satu", "Subscriber Kedua : dua", "Subscriber Kedua : tiga", "selesai".

```
Task :MainKt.main()
Subscriber Pertama : satu
Subscriber Pertama : dua
Subscriber Pertama : tiga
Subscriber Kedua : satu
Subscriber Kedua : dua
Subscriber Kedua : tiga
selesai
BUILD SUCCESSFUL in 1s
2 actionable tasks: 1 executed, 1 up-to-date
2:13:06 PM: Task execution finished 'MainKt.main()'.
```

14. Kesimpulannya, jika kita membuat suatu instance dari `Observable` dengan method `just` seperti kode yang telah ditulis, maka dia akan memancarkan (meng-emit) event apa saja?

Jawaban :

Meng-emit semua event yang ada didalamnya dan yang memanggilnya