Block 10 – Navigation mit Jetpack (Navigation II)

- Nachteile der Navigation über Intents
- Activities vs. Fragments
- Jetpack Navigation
  - Fragmente
  - Navigation Graph
  - Aufruf (ohne Parameter/ohne Rückgabe)
  - Aufruf (mit Parameter/ohne Rückgabe)
  - Aufruf (ohne Parameter/mit Rückgabe)
  - Aufruf (mit Parameter/mit Rückgabe)
  - Aufruf (mit Parameterobj./ohne Rückgabe)
  - Nachteile der Navigation mit Jetpack
  - ViewModelProvider u. CustomViewModel
- Back Stack
  - Standardverhalten
  - PopUpTo (Gleicher Kontext)
  - PopUpTo (Unterschiedlicher Kontext)
  - Single Top

Block 10 – Lernziele

In diesem Block werden Sie lernen . . .

- wie sich dynamische von statischer Navigation unterscheidet.
- wie Fragments analog zu Actitivities eingesetzt werden können.
- mit welchen Mitteln die statische Navigation einfacher realisiert werden kann.
- wie Argumente (Parameter) und Rückgabewerte typsicher übergeben werden können.
- wie sich der Back Stack durch den Aufruf/das Verlassen von Fragments auf- und abbaut.
- wie Daten zwischen Fragmentaufrufen bewahrt werden können.

10.1 Nachteile der Navigation über Intents (Aufruf)

Die Navigation zwischen Activities mit Hilfe von Intents ist ggf. sehr fehleranfällig, da das Aurufziel nur eine gültige Klasse sein muss und nicht weiter geprüft wird, ob Argumente (Parameter) und Rückgaben zwischen Aufrufer u. Augerufenem zusammen passen.

Die Navigation kann auch komplett *dynamisch* d. h. zur Laufzeit festgelegt werden:

```
val intent = Intent(applicationContext,
  Activity_noParams_noResult::class.java)
```

kann durch die Zeilen

```
val clazz  = Class.forName(
  "com.example.navigationi.
  Activity_noParams_noResult")
val intent = Intent(applicationContext, clazz)
```

ersetzt werden.

10.1 Nachteile der Navigation über Intents (Parameter u. Rückgabecodes

Weiterhin müsssen die Parameter als Key-Value-Paare in das `Extra` des `Intents` eingetragen werden, und bei dem Aurufziel wieder entnommen werden.

Bei der Rückgabe von Argumenten müssen ggf. in der aufrufenden Activity der `RequestCode` und der `Returncode` ausgewertet werden (um die Rückgabe einem Aufruf zuordnen zu können).

Die *dynamische* Navigation ist zwar sehr flexibel, jedoch ist es oftmals angemessener eine *statische* Navigation zu modellieren (d. h. die Navigationspfade bereits zur Entwicklungszeit festzulegen).

10.2 Activities vs. Fragments

Bisher wurden die mobilen Anwendungen ausschließlich durch Activities (Vgl. Sitemap) strukturiert.

Fragments bieten eine ergänzende Form eine grafische Oberfläche (GUI) und auch die Logik zu definieren und die mobile Anwendung so weiter zu modularisieren.

Mehrere Fragments können innerhalb von Activities genutzt werden. Dies ist besonders nützlich, falls bestimmte Inhalte nicht in jedem Kontext der Activity angezeigt werden sollen oder Activities bspw. auf unterschiedlichen Geräten (Smartphone/Tablet) angezeigt werden.

Fragments können nicht ohne eine umgebende Activity erzeugt werden. Sie müssen im Gegensatz zu Activities jedoch nicht in die `Manifest.xml` eingetragen werden - haben jedoch auch einen eigenen Lebenszyklus.

Sehr häufig werden mobile Anwendungen ausschließlich aus Fragments erzeugt.

10.3 Jetpack Navigation (Konfiguration)

Jetpack Navigation ist ein Rahmenwerk (`androidx.navigation`), welches den Entwickler bei der Navigation zwischen Activites und Fragments unterstützt.

In der modulspezifischen Gradle-Datei `build.gradle` (app `build.gradle`) sind bei den `dependencies{...}` folgende Einträge zu ergänzen:

```
implementation
↩'androidx.navigation:navigation-fragment-ktx:2.5.3'
implementation
↩'androidx.navigation:navigation-ui-ktx:2.5.3'
```

Im Bereich `plugins{...}` ist folgender Eintrag zu ergänzen:

```
id 'androidx.navigation.safeargs'
```

10.3 Jetpack Navigation (Konfiguration - Forts.)

In der Gradle-Datei `build.gradle` (`build.gradle`) des Projekts muss in dem Bereich `buildscript{... dependencies{...}}` noch folgender Eintrag hinzugefügt werden:

```
classpath('androidx.navigation:
↩navigation-safe-args-gradle-plugin:2.5.3')
```

Die Einbindung von `Safeargs` erlaubt die Verwendung sicherer Argument bei der Navigation d. h. dem gegenseitigen Aufruf von Fragements.

10.3.1 Fragmente

Die Fragmente bilden später die Knoten (Navigationsziele) im Navigation Graph. Analog zu der Navigation mit Activities wird für jede benötigte Ansicht ein `Fragment` angelegt:

Über File ▷ New ▷ Fragment ▷ Fragment (Blank) kann ein neues Fragment erstellt werden (z. B. Klasse `FragmentMain.kt` i. v. m. Layout `fragment_main.xml`).

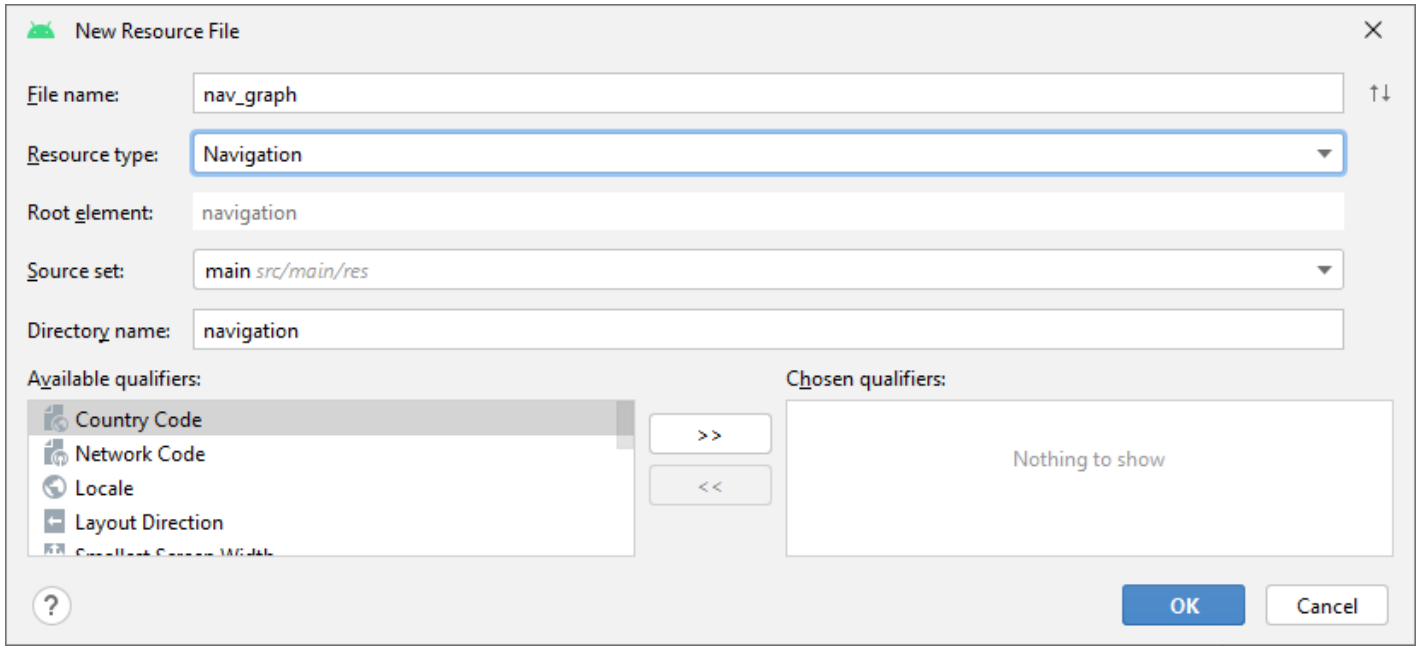
Die Layouts der Fragments werden mit den bereits bekannten Layout-Typen (Linear Layout, Frame Layout, Constrain Layout etc.) angelegt.

Folgend sind alle Fragments der geplanten mobilen Anwendung zu erstellen (die Logik wird dabei noch nicht implementiert).

10.3.2 Navigation Graph (Anlegen)

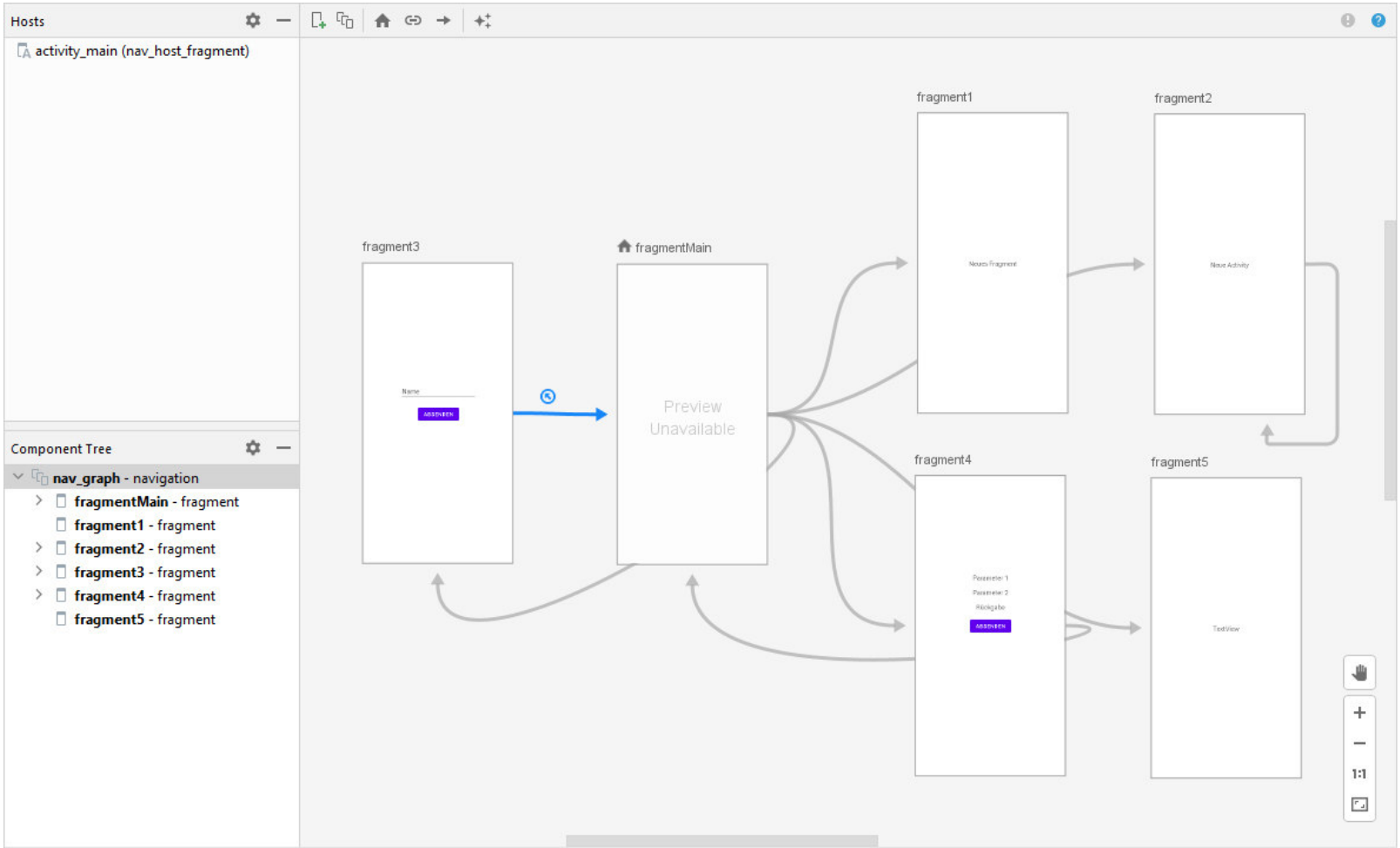
Anschließend wird über das Kontextmenü (Rechtsklick auf den Ordner res) die Option New ▷ Android Resource File der Navigation Graph angelegt.

In dem folgenden Dialog ist als Resource type: “Navigation” auszuwählen. Die resultierende Datei kann z. B. `nav_graph` benannt werden.



10.3.2 Navigation Graph (Knoten)

Über die Hinzufügen-Schaltfläche können die Fragments in den Navigation Graph als Knoten (Navigationsziele) eingefügt werden:



10.3.2 Navigation Graph (Startknoten)

Ein Fragment muss als Startknoten `app:startDestination` ausgezeichnet werden (Haus-Symbol).

```
<navigation xmlns:android=...
    ...
    app:startDestination="@id/fragmentMain">
<fragment
    android:id="@+id/fragmentMain"
    android:name="com.example.navigationii.FragmentMain"
    android:label="FragmentMain">
... </fragment>
</navigation>
```

Wird kein Startknoten `app:startDestination` definiert, führt dies zu einem Absturz der mobilen Anwendung (`no start destination defined`).

10.3.2 Navigation Graph (Kanten)

Die Kanten des Navigation Graph (Actions) realisieren die Navigationsmöglichkeiten zwischen den Knoten (Fragments).

Dabei werden in dem GUI-Editor an den Ankerpunkten Verbindungen zwischen den Knoten hergestellt (ggf. auch reflexiv).

```
<navigation xmlns:android=...>
<fragment android:id="@+id/fragmentMain" ... >
<action
    android:id="@+id/action_fragmentMain_to_fragment1"
    app:destination="@id/fragment1" />
... </fragment>
<fragment android:id="@+id/fragment1" ... />
</navigation>
```

Die Actions definieren die generelle Navigationsmöglichkeit von einem zu einem anderen Fragment.

Actions sind damit zunächst keinen GUI-Elementen der Fragment-Layouts zugeordnet, dies erfolgt später programmatisch.

10.3.2 Navigation Graph (Argumente)

Einem `Fragment` (Knoten) können Argumente übergeben werden. Über die Editor des Navigation Graph können diese angelegt/gelöscht werden, und ergeben einen korrespondierenden Eintrag in der Layout-Datei:

```
<fragment android:id="@+id/fragment2" ...>
    <argument
        android:name="parameter"
        app:argType="string" />
...
</fragment>
```

Über die Attribute `android:defaultValue` und `app:nullable` kann ein Standardwert oder die mögliche Belegung mit `null` eingestellt werden.

10.3.2 Navigation Graph (Aufrufeinstellungen)

Ohne weitere Einstellungen bildet sich ein Backstack (analog zu dem Activitystack bei der Nutzung von Activities).

Über die Attribute `app:popUpTo` bzw. `app:popUpToInclusive` kann der Backstack wieder abgebaut werden, z. B. wenn zur Hauptactivity zurückgekehrt werden soll.

```
<navigation xmlns:android=...>
<fragment android:id="@+id/fragment3" ... >
<action
    android:id="@+id/action_fragment3_to_fragmentMain"
    app:destination="@id/fragmentMain"
    app:popUpTo="@id/fragmentMain"
    app:popUpToInclusive="true" />
... </fragment>
<fragment android:id="@+id/fragmentMain" ... />
</navigation>
```

Hinweis: In diesem Fall wird das Fragment (MainFragment) neu erzeugt und nicht wieder in den Vordergrund gebracht.

10.3.2 Navigation Graph (Referenzierung)

Der fertig angelegte Navigation Graph kann noch nicht verwendet werden, da er nirgends in der mobilen Anwendung genutzt wird. Zur Nutzung wird daher eine Activity angelegt, die den Navigation Graph referenziert.

Anlegen einer neuen Activity (`MainActivity.kt`) mit dem folgenden Layout (`activity_main.xml`):

```
<androidx.constraintlayout.widget.ConstraintLayout ...>
<androidx.fragment.app.FragmentContainerView
    android:id="@+id/nav_host_fragment"
    android:name="androidx.navigation.fragment.NavHostFragment"
    app:defaultNavHost="true"
    app:navGraph="@navigation/nav_graph"
    ... />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Die View `FragmentContainerView` referenziert über das Attribut `app:navGraph` den zuvor angelegten Navigation Graph. In die `FragmentContainerView` wird bei der Ausführung der mobilen Anwendung der Startknoten des Navigation Graphs geladen.

10.3.2 Navigation Graph (Standardnavigation)

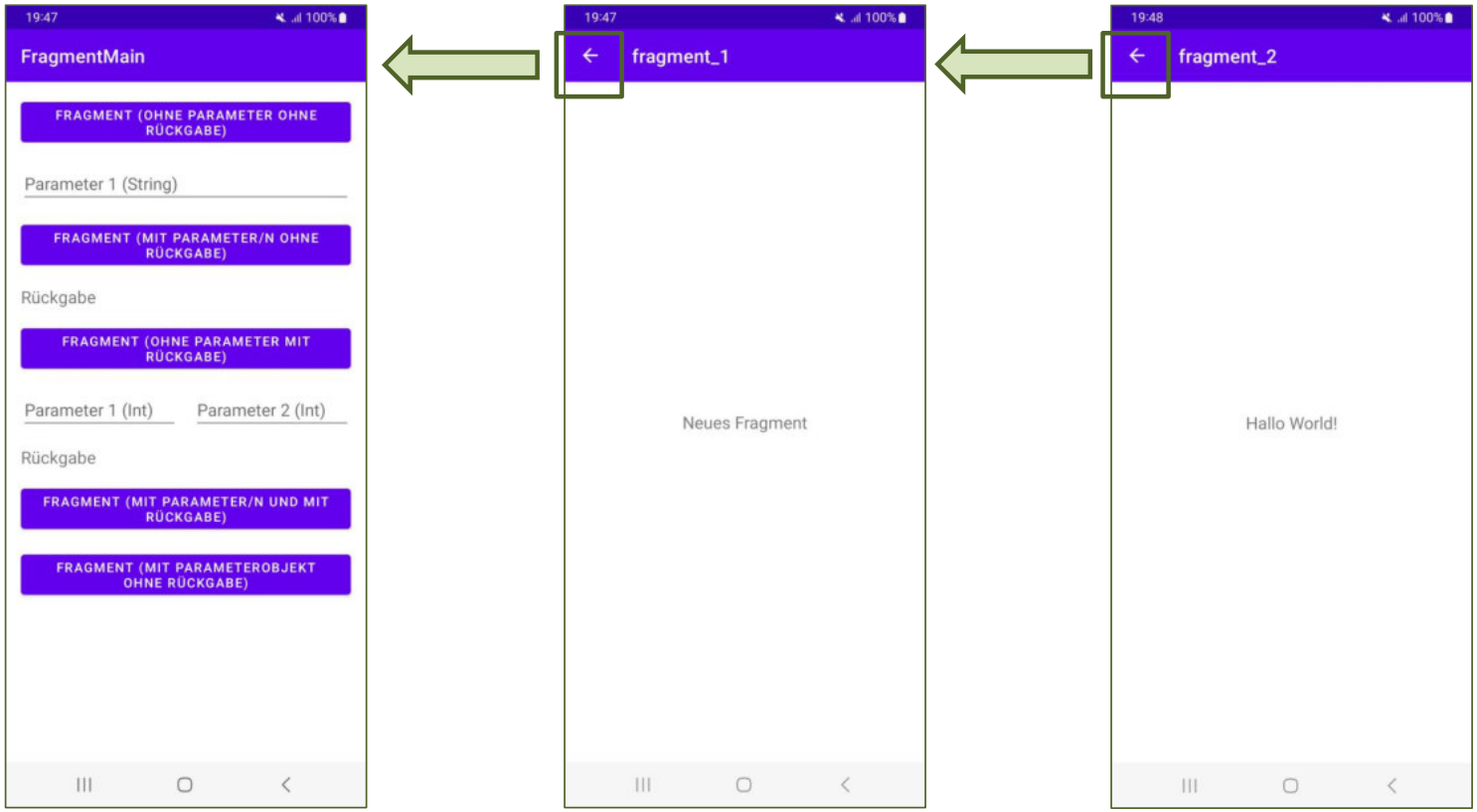
In der Klasse `MainActivity.kt` wird über die Referenz auf die `FragmentContainView` der `NavController` bezogen und die `ActionBar` konfiguriert.

```
val navHostFragment = supportFragmentManager.
  findFragmentById(R.id.nav_host_fragment) as NavHostFragment

navController = navHostFragment.navController
navController.enableOnBackPressed(true)
appBarConfiguration = AppBarConfiguration.Builder(
  R.id.fragmentMain
).build()

NavigationUI.setupActionBarWithNavController(
  this, navController, appBarConfiguration)
```

Die Standardnaviagtion ermöglicht insbesondere eine Zurück-Navigation (`enableOnBackPressed`), auch wenn in dem Navigation Graph keine Action (Kante) definiert wurde.

10.3.2 Standardnavigation (Beispiel)

Standardnavigation (Zurück) über die `ActionBar`:



10.3.3 Aufruf (ohne Parameter/ohne Rückgabe)

Nach der Speicherung des Navigation Graphs wird für alle Fragments (Knoten) die miteinander verbunden wurden jeweils eine Navigationklasse generiert.

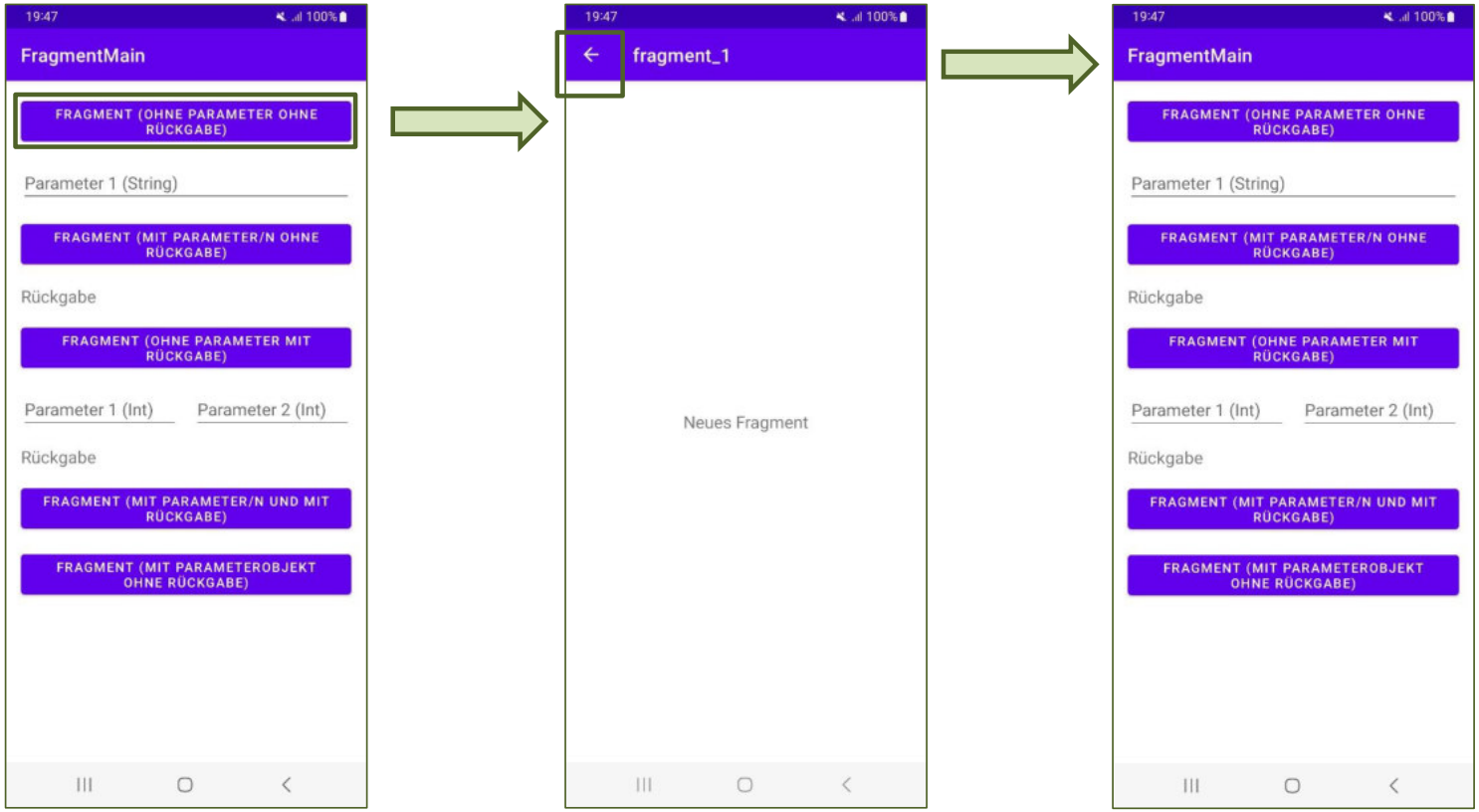
Beispiel: Für die Klassen `FragmentMain` u. `Fragment1` wird die Klasse `FragmentMainDirections` mit der Methode `actionFragmentMainToFragment1()` generiert.

Ein Navigationsaufruf ohne Argument sieht dann wie folgt aus:

```
button1.setOnClickListener(object : View.OnClickListener {
  override fun onClick(p0: View?) {
    val action = FragmentMainDirections.
      actionFragmentMainToFragment1()
    findNavController().navigate(action)
  }
})
```

Die Navigation ist damit statisch zur Entwicklungszeit abgesichert, das Navigationsziel ist vorhanden.

10.3.3 Aufruf (ohne Parameter/ohne Rückgabe) - Beispiel



Backstack:

1. `[FragmentNavigation, FragmentMain]`
2. `[FragmentNavigation, FragmentMain, fragment_1]`
3. `[FragmentNavigation, FragmentMain]`

10.3.4 Aufruf (mit Parameter/ohne Rückgabe)

Aufruf eines Fragments mit Parameter ohne Rückgabewert:

```
button2.setOnClickListener(object : View.OnClickListener {
  override fun onClick(p0: View?) {
    val action = FragmentMainDirections.
      actionFragmentMainToFragment2(
        editTextParameter11.text.toString())
    findNavController().navigate(action)
  }
})
```

Hier wird das Argument direkt als Argument der Methode `actionFragmentMainToFragment2()` übergeben.

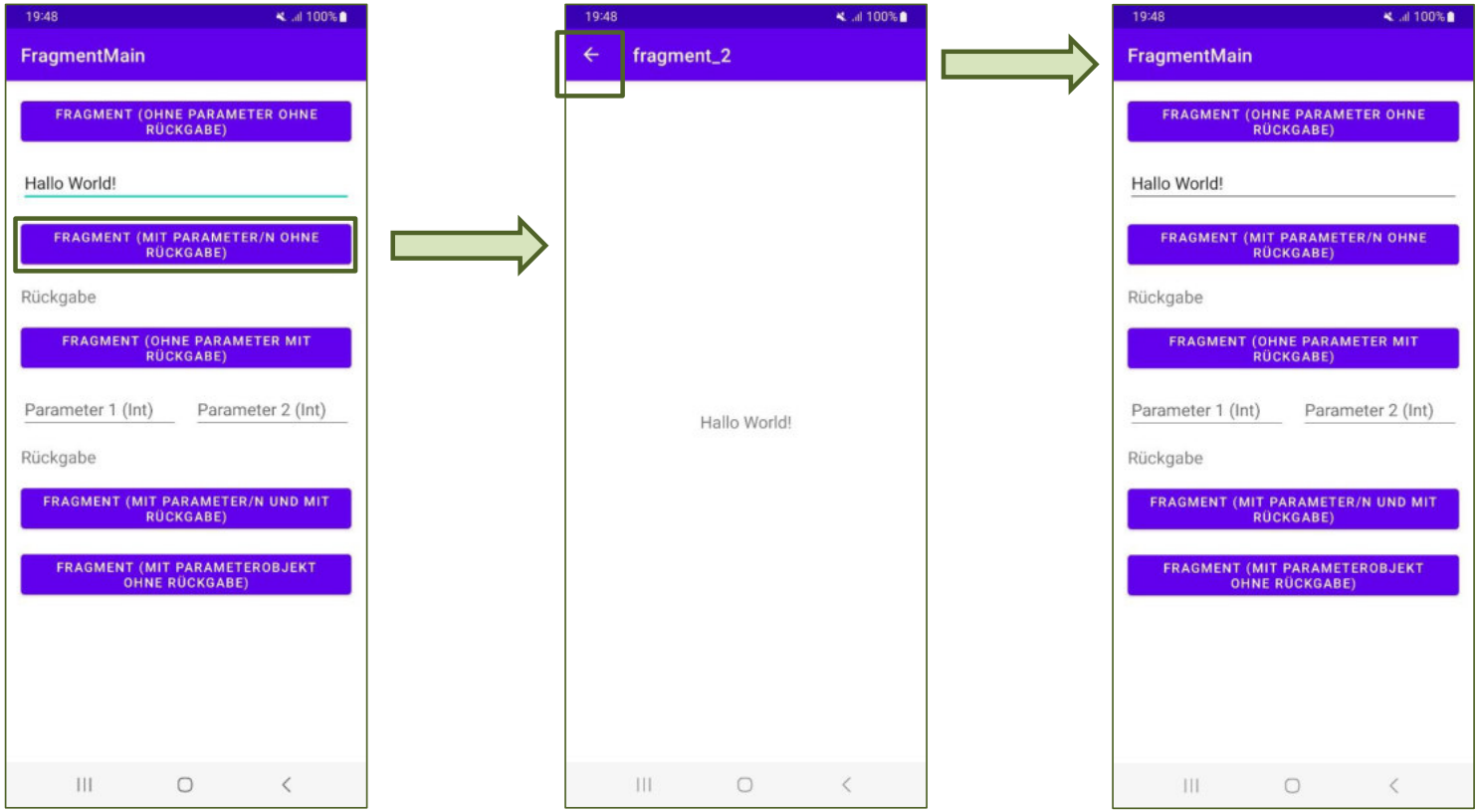
10.3.4 Aufruf (mit Parameter/ohne Rückgabe)

Aufruf eines Fragments mit Parameter ohne Rückgabewert:

```
val args : Fragment2Args by navArgs()
...
override fun onViewCreated(view: View,
  savedInstanceState: Bundle?) {
  super.onViewCreated(view, savedInstanceState)
  textView = view.findViewById(R.id.textView2)
  textView.setText(args.parameter)
  ...
}
```

Das Argument wird über den Parameternamen (`parameter`) aus dem Objekt `args` (der aus dem Navigation Graph generierten Klasse `Fragment2Args`) entnommen.

10.3.4 Aufruf (mit Parameter/ohne Rückgabe) - Beispiel



Backstack:

1. `[FragmentNavigation, FragmentMain]`
2. `[FragmentNavigation, FragmentMain, fragment_2]`
3. `[FragmentNavigation, FragmentMain]`

10.3.5 Aufruf (ohne Parameter/mit Rückgabe)

Aufruf eines Fragments ohne Parameter mit Rückgabewert:

```
button3.setOnClickListener(object : View.OnClickListener {
  override fun onClick(p0: View?) {
    val action = FragmentMainDirections.
      actionFragmentMainToFragment3()
    findNavController().navigate(action)
  }
})
```

Hier ist in dem aufrufenden Fragment nichts weiter zu implementieren (z. B. auch keine `ActivityResultLauncher` wie bei den Activities).

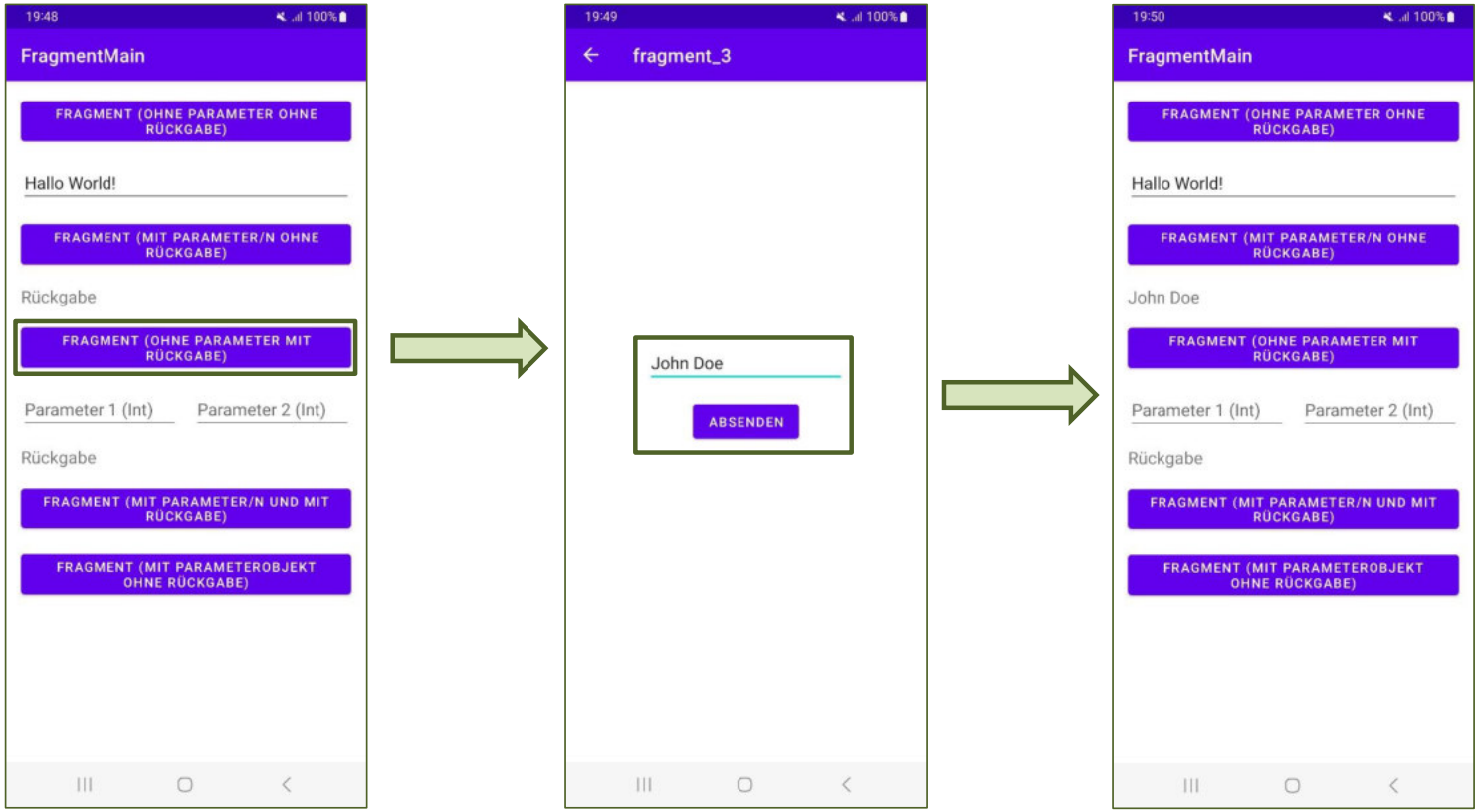
10.3.5 Aufruf (ohne Parameter/mit Rückgabe)

Aufruf eines Fragments ohne Parameter mit Rückgabewert:

```
val args : FragmentMainArgs by navArgs()
...
override fun onViewCreated(view: View,
  savedInstanceState: Bundle?) {
  super.onViewCreated(view, savedInstanceState)
  textViewReturn11 = view
    .findViewById<TextView>(R.id.textViewReturn11)
  textViewReturn.setText(args.name)
  ...
}
```

Das Argument (hier Rückgabewert) wird über den Parameternamen (`name`) aus dem Objekt `args` (der aus dem Navigation Graph generierten Klasse `FragmentMainArgs`) entnommen.

10.3.5 Aufruf (ohne Parameter/mit Rückgabe) - Beispiel



Backstack:

1. `[FragmentNavigation, FragmentMain]`
2. `[FragmentNavigation, FragmentMain, fragment_3]`
3. `[FragmentNavigation, FragmentMain]`

10.3.6 Aufruf (mit Parameter/mit Rückgabe)

Aufruf eines Fragments mit Parameter mit Rückgabewert:

```
button4.setOnClickListener(object : View.OnClickListener {
  override fun onClick(p0: View?) {
    ...
    val action = FragmentMainDirections.
      actionFragmentMainToFragment4(
        parameter1, parameter2)
    findNavController().navigate(action)
    ...
  }
})
```

Hier werden die Argumente direkt als Argumente der Methode `actionFragmentMainToFragment4()` übergeben.

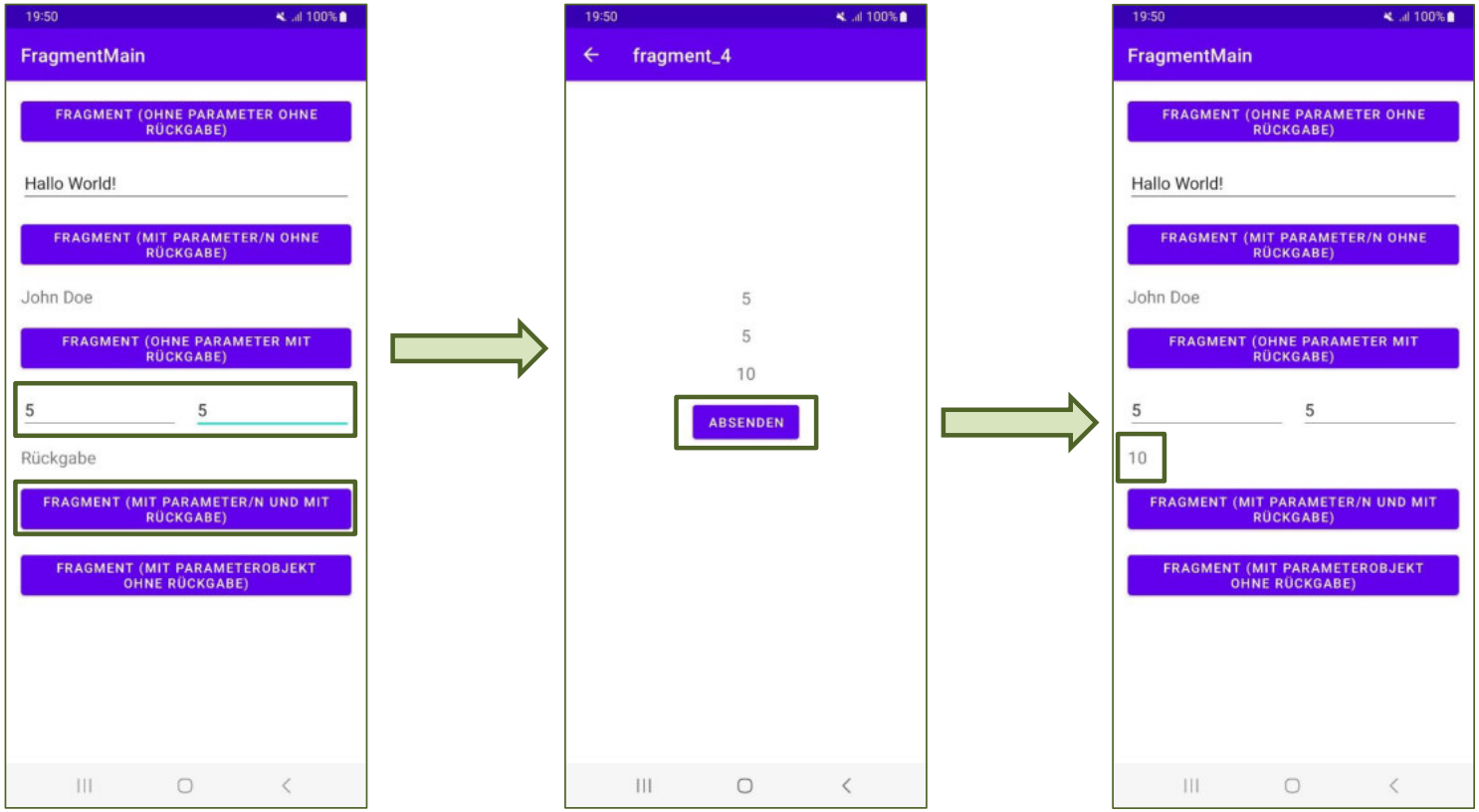
10.3.6 Aufruf (mit Parameter/mit Rückgabe)

Aufruf eines Fragments mit Parameter mit Rückgabewert:

```
val args : FragmentMainArgs by navArgs()
...
override fun onViewCreated(view: View,
  savedInstanceState: Bundle?) {
  super.onViewCreated(view, savedInstanceState)
  textViewReturn12 =
    view.findViewById<TextView>(R.id.textViewReturn12)
  textViewReturn12.setText(args.ergebnis.toString())
  ...
}
```

Das Argument (hier Rückgabewert) wird über den Parameternamen (`ergebnis`) aus dem Objekt `args` (der aus dem Navigation Graph generierten Klasse `FragmentMainArgs`) entnommen.

10.3.6 Aufruf (mit Parameter/mit Rückgabe) - Beispiel



Backstack:

1. `[FragmentNavigation, FragmentMain]`
2. `[FragmentNavigation, FragmentMain, fragment_4]`
3. `[FragmentNavigation, FragmentMain]`

10.3.7 Aufruf (mit Parameterobj./ohne Rückgabe)

Aufruf eines Fragments mit Parameterobjekt ohne Rückgabewert:

```
button5.setOnClickListener(object : View.OnClickListener {
  override fun onClick(p0: View?) {
    ...
    val person : Person = Person ("John", "Doe")
    val action = FragmentMainDirections.
      actionFragmentMainToFragment5(person)
    findNavController().navigate(action)
    ...
  }
})
```

Hier werden die Argumente direkt als Argumente der Methode `actionFragmentMainToFragment5()` übergeben. Dabei kann es sich auch um komplexe Typen handeln.

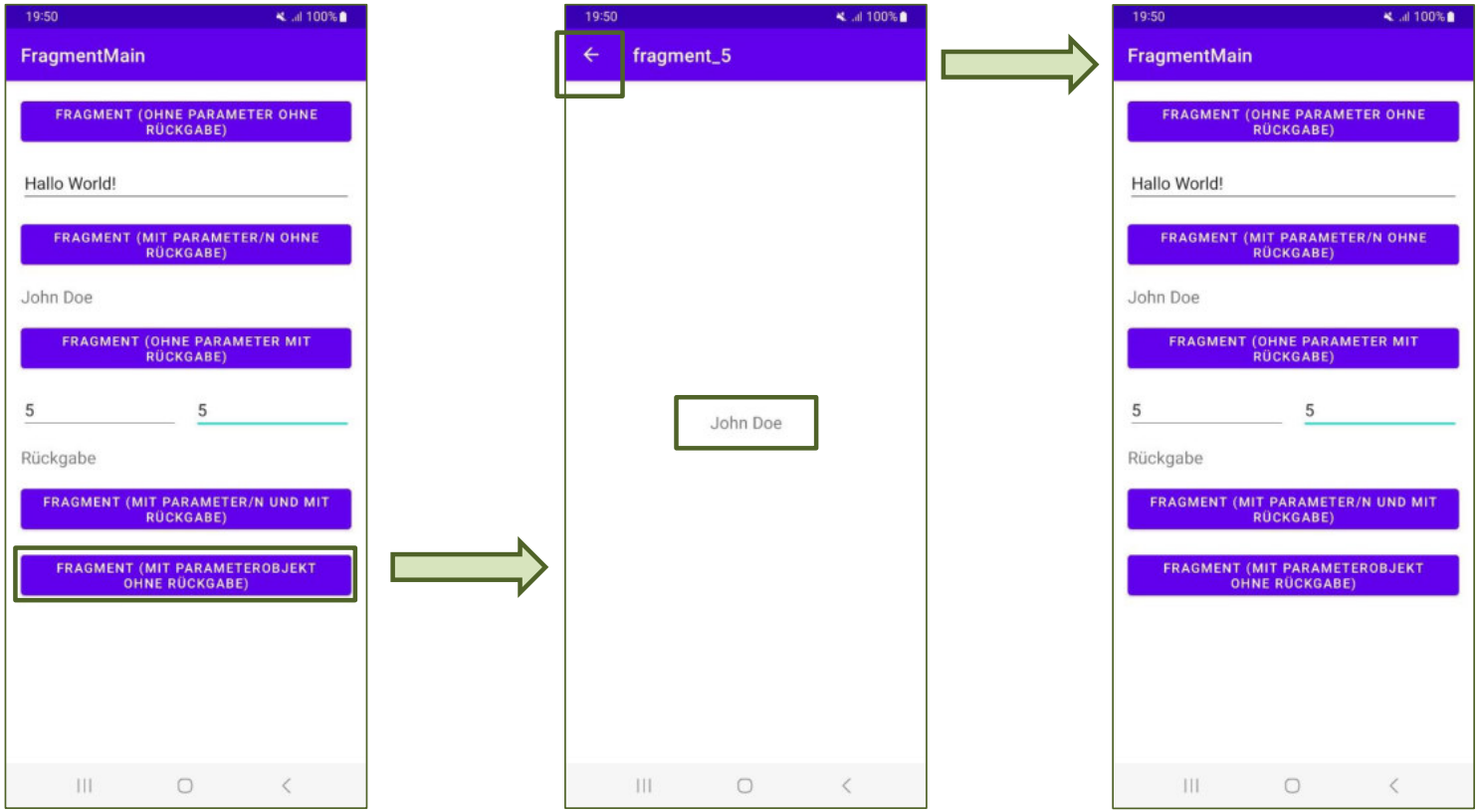
10.3.7 Aufruf (mit Parameterobj./ohne Rückgabe)

Aufruf eines Fragments mit Parameterobjekt ohne Rückgabewert:

```
val args : Fragment5Args by navArgs()
...
override fun onViewCreated(view: View,
  savedInstanceState: Bundle?) {
  super.onViewCreated(view, savedInstanceState)
  val textView = view.findViewById<TextView>(R.id.textView6)
  textView.setText(args.person.vorname
    + " " +  args.person.nachname)
  ...
}
```

Das Argument wird über den Parameternamen (`person`) aus dem Objekt `args` (der aus dem Navigation Graph generierten Klasse `Fragment5Args`) entnommen. Dabei können die getter-Methoden für die einzelnen Attribute des Objekts genutzt werden.

10.3.7 Aufruf (mit Parameterobj./ohne Rückgabe) - Beispiel



Backstack:

1. `[FragmentNavigation, FragmentMain]`
2. `[FragmentNavigation, FragmentMain, fragment_5]`
3. `[FragmentNavigation, FragmentMain]`

10.3.8 Nachteile der Navigation mit Jetpack

- Es scheint keine einfache Lösung zu geben, im Navigationsstack mit Rückgabewerten zurückzugehen. Der Aufruf von Knoten führt zur neuen Instanzierung des Knotens.
- Wird ein Fragment (Knoten) in mehreren Kontexten genutzt (d. h. mehrere eigehende Aufrufe), dann müssen alle Aufrufer immer die gesamten Argumente übermitteln, unabhängig ob diese dem Aufrufer bekannt sind
- Die Argumente können zwar `null` sein, aber dann gehen möglicherweise Wertbelegungen aus früheren Aufrufen verloren.
- Es gibt zunächst keine direkte Möglichkeit festzustellen welcher Aufrufer einen Aufruf gemacht hat (z. B. `requestCode`) bzw. bei einer Rückgabe von welchem Fragement die Rückgabe erfolgt (z. B. `returnCode`).

10.3.9 ViewModelProvider u. CustomViewModel

Aufgrund der Nachteile der Navigation mit Jetpack wird ein `CustomViewModel` genutzt, um die Argumente nach bei dem Wiederaufruf eines Fragments (Knotens) wiederhezustellen.

Die Methode `onSaveInstanceState()` sichert bei dem Zerstören eines Fragments (`onDestroy()`) alle Werte in einem `CustomViewModel` (erzeugt über ein `ViewModelProvider`).

Bei dem Durchlaufen der Methode `onViewCreated()` werden alle Werte - sofern vorhanden - aus dem `CustomViewModel` ausgelesen.

10.4 Backstack (Funktion)

Zur Betrachtung des Backstacks wird eine Hilfsmethode (`printBackStack()`) genutzt:

```
fun printBackStack(navController: NavController, tag : String) {

  val bse_deque : ArrayDeque<NavBackStackEntry> =
    navController.backQueue

  val n : Int = bse_deque.size-1
  var backStack_String : String = "BackStack ["

  for (i in 0..n){
    val nbse : NavBackStackEntry = bse_deque.get(i)
    backStack_String = backStack_String + nbse.destination.label
      + " (" + nbse +") "+ ", "
  }

  ...
  backStack_String = backStack_String + "]"
  Log.e(this.javaClass.name + " (" + tag + ")", backStack_String)

}
```

10.4 Backstack (Ausgabe)

Ausgabe (Beispiel):

```
1. BackStack
[FragmentNavigation (....NavBackStackEntry@fe46e339),
FragmentMain (....NavBackStackEntry@4727536d)]
2. BackStack
[FragmentNavigation (....NavBackStackEntry@fe46e339),
FragmentMain (....NavBackStackEntry@4727536d),
fragment\_3 (....NavBackStackEntry@2b538f8d)]
3. BackStack
[FragmentNavigation (....NavBackStackEntry@fe46e339),
FragmentMain (....NavBackStackEntry@19f127e6)]
```

10.4.1 Standardverhalten

Der Aufruf mit der Action `Fragment2Directions.actionFragment2Self(‘‘Task A’’)` leitet einen (einfachen) rekursiven Aufruf des Knotens (Fragment) ein.

```
if (textView.text.toString().equals("Task A")) {
    val action = Fragment2Directions.actionFragment2Self("Task A1")
    // FragmentNavigation -> FragmentMain
    // -> Fragment2 (A) -> Fragment2 (A1)
    findNavController().navigate(action)
}
```

10.4.1 Standardverhalten

Rekursiver Aufruf eines Fragments (Standardverhalten):

Backstack:

1. `[FragmentNavigation, FragmentMain]`
2. `[FragmentNavigation, FragmentMain, fragment_2, fragment_2]`
3. `[FragmentNavigation, FragmentMain, fragment_2]`
4. `[FragmentNavigation, FragmentMain]`

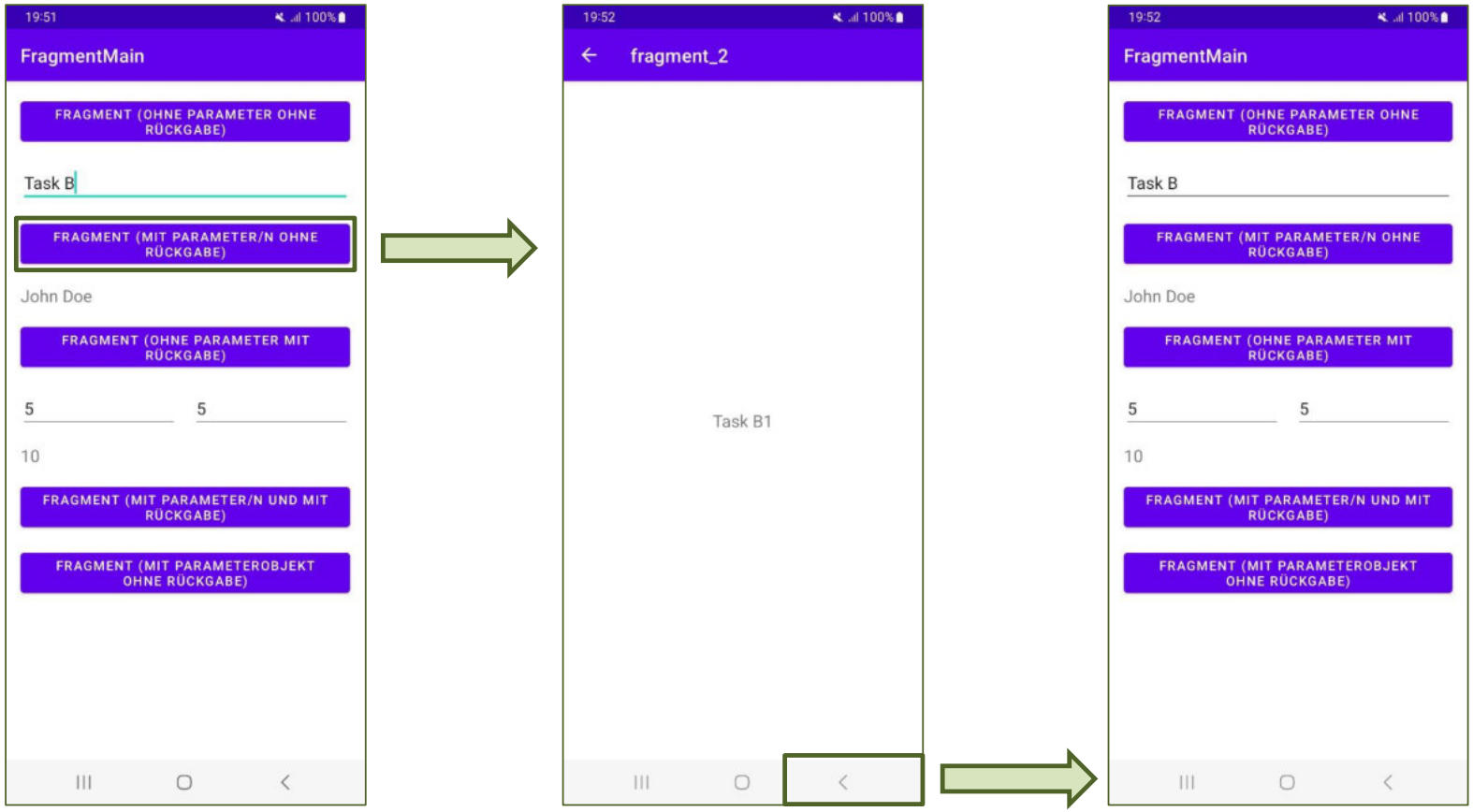
10.4.2 PopUpTo (Gleicher Kontext)

Der Aufruf mit der Action `Fragment2Directions.actionFragment2Self(‘‘Task B’’)` leitet einen (einfachen) rekursiven Aufruf des Knotens (Fragment) ein. Dabei werden alle Knoten (Fragmente) bis zum Fragment `R.id.fragmentMain` (exklusiv) entfernt (auch Fragment2 (B)).

```
if (textView.text.toString().equals("Task B")) {
    val action = Fragment2Directions.actionFragment2Self("Task B1")
    val navoptions = NavOptions.Builder()
        .setPopUpTo(R.id.fragmentMain, false).build()
    // FragmentNavigation -> FragmentMain -> Fragment2 (B1)
    findNavController().navigate(action, navoptions)
}
```

10.4.2 PopUpTo (Gleicher Kontext)

Rekursiver Aufruf eines Fragments (PopUpTo):



Backstack:

1. `[FragmentNavigation, FragmentMain]`
2. `[FragmentNavigation, FragmentMain, fragment_2]`
3. `[FragmentNavigation, FragmentMain]`

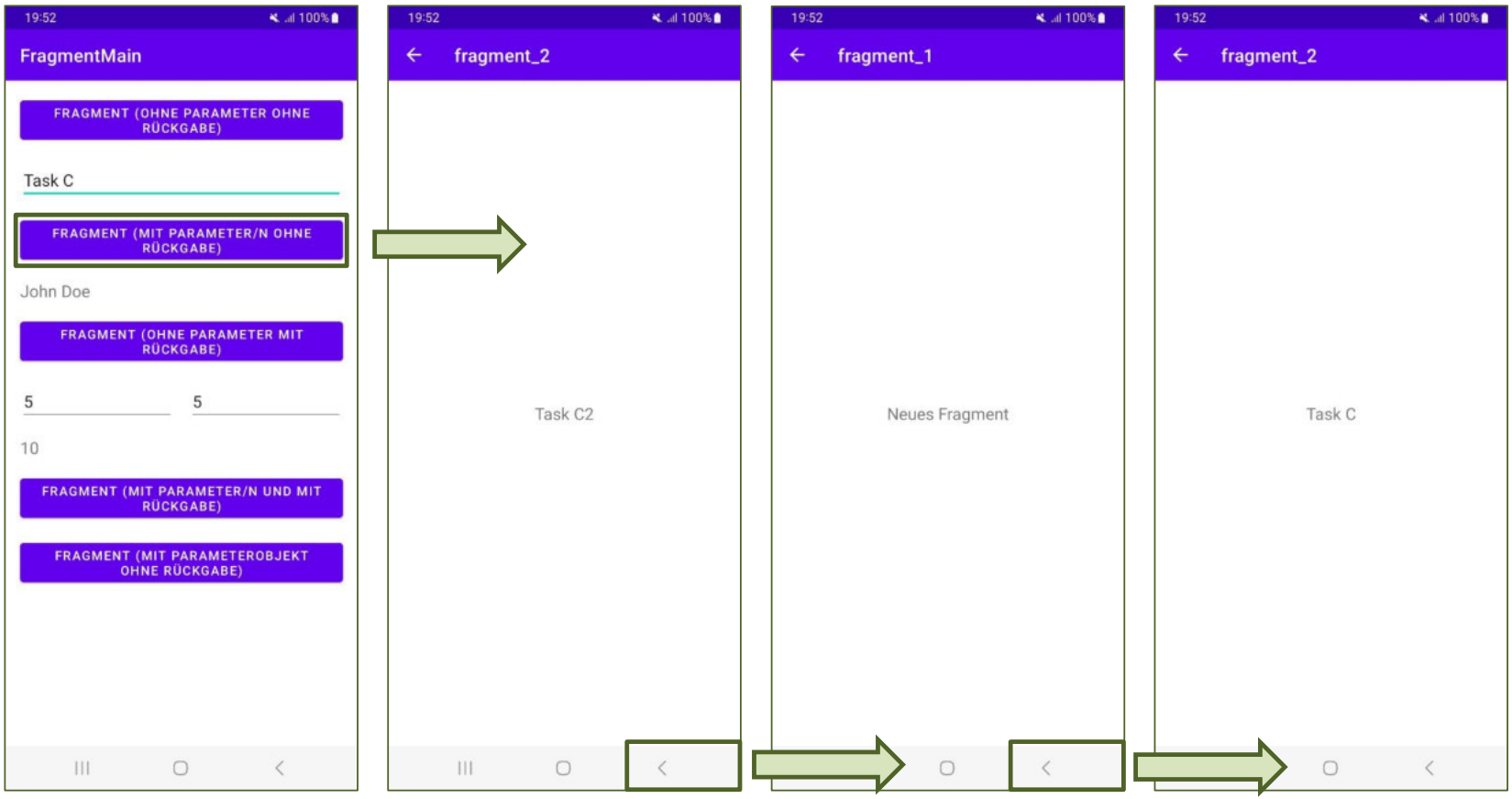
10.4.3 PopUpTo (Unterschiedlicher Kontext)

Der Aufruf mit der Action `Fragment2Directions.actionFragment2Self(‘‘Task C’’)` leitet einen (zweifachen) rekursiven Aufruf des Knotens (Fragment) ein. Dabei werden alle Knoten (Fragmente) bis zum Fragment `R.id.fragment1` (exklusiv) entfernt (auch ein nachfolgendes Fragment2 (C1)).

```
if (textView.text.toString().equals("Task C")) {
  findNavController().navigate(R.id.fragment1)
  val navoptions = NavOptions.Builder()
    .setPopUpTo(R.id.fragment1, false).build()
  val args = Bundle()
  args.putString("parameter", "Task C1")
  findNavController().navigate(R.id.fragment2, args, navoptions)
}

if (textView.text.toString().equals("Task C1")) {
  val navoptions = NavOptions.Builder()
    .setPopUpTo(R.id.fragment1, false).build()
  val args = Bundle()
  args.putString("parameter", "Task C2")
  findNavController().navigate(R.id.fragment2, args, navoptions)
}
```

10.4.3 PopUpTo (Gleicher Kontext)

Rekursiver Aufruf eines Fragments (PopUpTo):



Backstack:

1. `[FragmentNavigation, FragmentMain]`
2. `[FragmentNavigation, FragmentMain, fragment_2, fragment_1, fragment_2]`
3. `[FragmentNavigation, FragmentMain, fragment_2, fragment_1]`
4. `[FragmentNavigation, FragmentMain, fragment_2]`

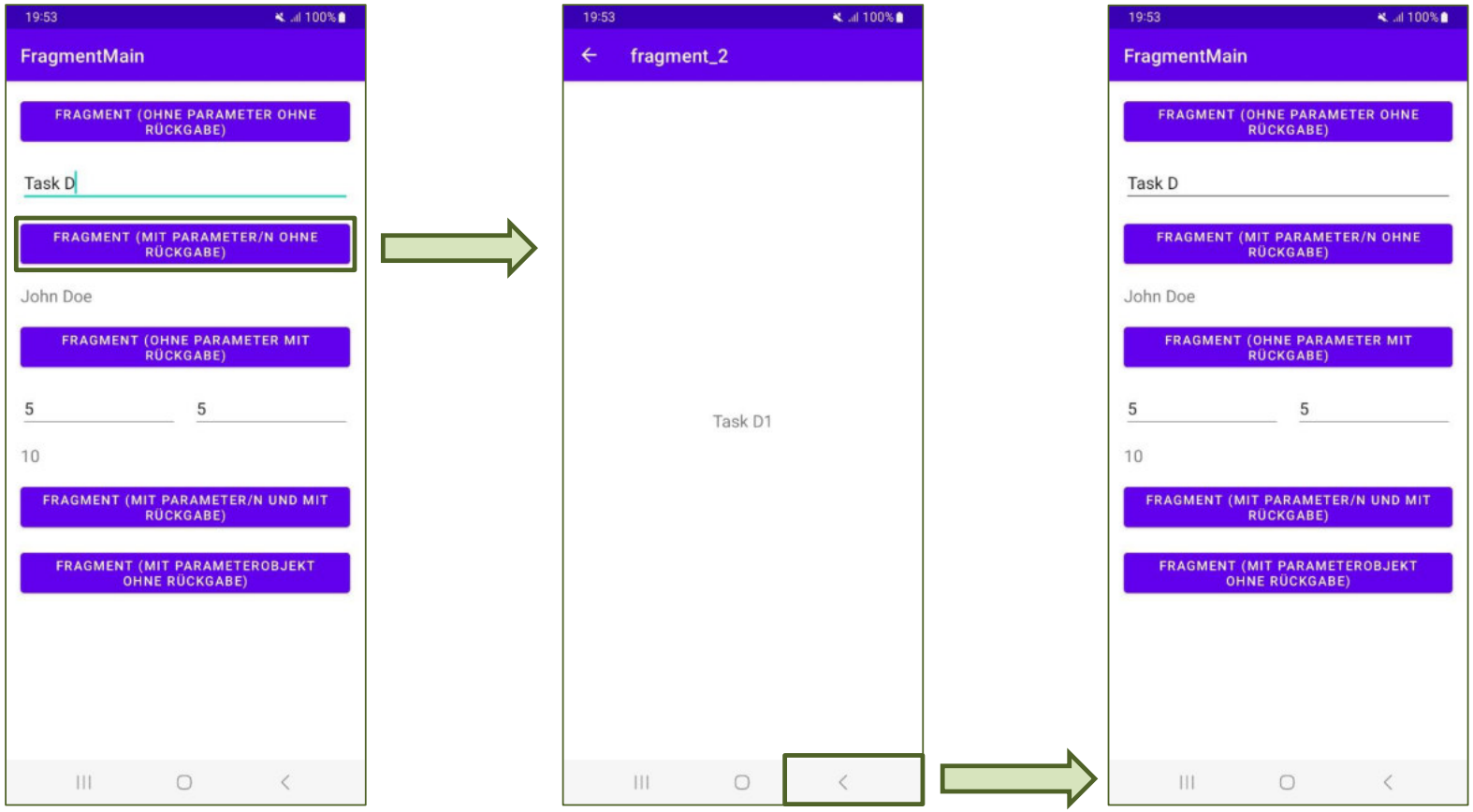
10.4.4 Single Top

Der Aufruf mit der Action `Fragment2Directions.actionFragment2Self(‘‘Task D’’)` leitet einen (einfachen) rekursiven Aufruf des Knotens (Fragment) ein. Dabei werden alle Knoten (Fragmente) desselben Typs entfernt.

```
if (textView.text.toString().equals("Task D")) {
  val action = Fragment2Directions.actionFragment2Self("Task D1")
  val navoptions : NavOptions = NavOptions.Builder()
    .setLaunchSingleTop(true).build()
  // FragmentNavigation, FragmentMain -> Fragment2 (D1)
  findNavController().navigate(action, navoptions)
}
```

10.4.4 Single Top

Rekursiver Aufruf eines Fragments (Single Top):



Backstack:

1. `[FragmentNavigation, FragmentMain]`
2. `[FragmentNavigation, FragmentMain, fragment_2]`
3. `[FragmentNavigation, FragmentMain]`

Block 10 – Zusammenfassung

- Die Modellierung eines Navigation Graphs erlaubt die statische Festlegung von Navigationspfaden inkl. der Argumente (Parameter).
- Aus dem Navigation Graph werden zur Entwicklungszeit Klassen generiert, die die Navigation ermöglichen.
- Die Parameterübergabe erfolgt in gewohnter Syntax in Form von geordneten Methodenargumenten.
- Über verschiedene Optionen kann der Aufbau des Backstacks gesteuert werden.
- Möglicherweise entgegen der Intuition führt die Rückkehr zu einem Fragment nicht zu “demselben”, sondern zu “dem gleichen” (Neuen) Fragment. Die Daten müssen über ein CustomViewModel in das neue Fragment übertragen werden.

Block 10 – Weitere Aufgaben

- Prüfen Sie im Programmcode zu diesem Block oder recherchieren Sie, ob und welche Zustände bzw. Methoden bei der Erzeugung und Zerstörung eines Fragments durchlaufen werden.
- Erzeugen Sie programmatisch (z. B. durch eine Schleife) eine hohe Anzahl von Fragments. Beobachten Sie das Verhalten des Systems.
- Testen Sie, ob das Layout eines Fragments kontextsensitiv bzgl. der Geräteorientierung (Portrait-/Landscape-Modus) ist.

Block 10 – Literatur I