Програмування мобільних пристроїв

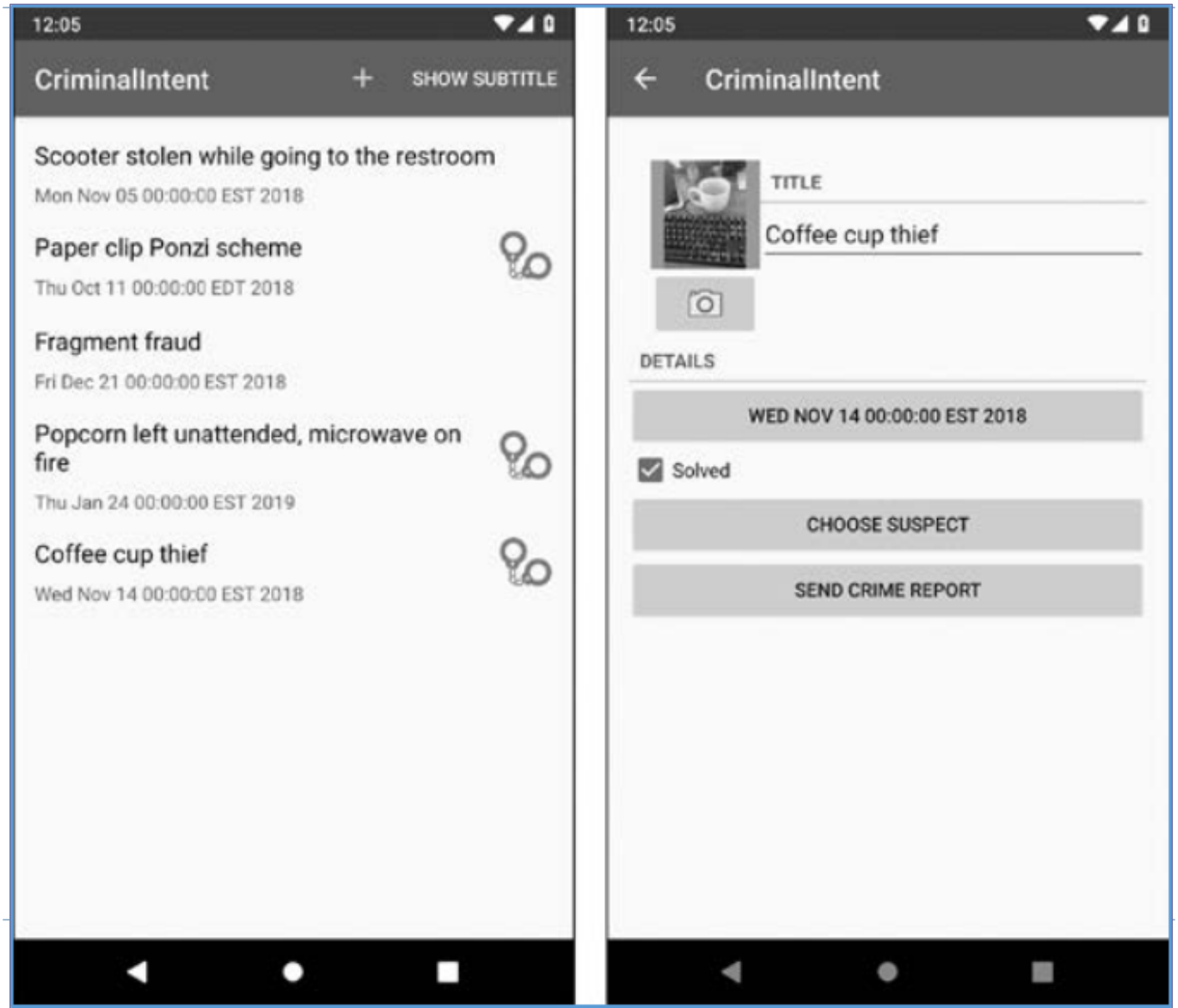
Використання фрагментів

Слайди до лекцій (6 змістовий модуль)

Функціональні вимоги до програми CriminalIntent

- Починаючи з цього розділу ми почнемо будувати програму CriminalIntent. Вона призначена для зберігання інформації про "офісні злочини": залишений у раковині брудний посуд, порожній лоток принтера після друку документів тощо.
- У програмі CriminalIntent користувач створює запис про злочин із заголовком, датою та фотографією. Також можна вибрати підозрюваного в адресній книзі та надіслати скаргу електронною поштою, опублікувати її в Twitter, Facebook або іншій програмі. Повідомивши про злочин, користувач звільняється від негативу і може зосередитися на поточному завданні.
- CriminalIntent – складний застосунок, у якому використовується інтерфейс типу "список/деталізація": на головному екрані виводиться список зареєстрованих злочинів. Користувач може додати новий або вибрати існуючий злочин для перегляду та редагування інформації.

Функціональні вимоги до програми CriminalIntent



Гнучкість інтерфейсу користувача

- Базуючись на досвіді розробки попереднього застосунку, можна спроєктувати застосунок типу "список/деталізація" з двох активностей: для керування списком та для керування деталізованим поданням. Така архітектура працює, але може знадобитися більш складна схема представлення інформації та навігації між екранами: наприклад, при роботі з програмою на планшеті. Екрани планшетів та деяких великих телефонів дозволяють одночасно відображати список та деталізацію (принаймні в альбомній орієнтації).
- Користувач переглядає опис злочину на телефоні та хоче побачити наступний злочин у списку. Було б зручно, якби користувач міг провести пальцем на екрані, щоб перейти до наступного злочину без повернення до списку. Такий сценарій передбачає гнучкість інтерфейсу користувача: можливість формування та зміни представлення активності під час виконання залежно від того, що потрібно користувачеві або пристрою.

Гнучкість інтерфейсу користувача

- Подібна гнучкість в активності не передбачена. Представлення активності можуть змінюватися під час виконання, але код, який керує цими змінами, повинен перебувати у представленні. В результаті активність тісно зв'язується з конкретним екраном, з яким працює користувач.

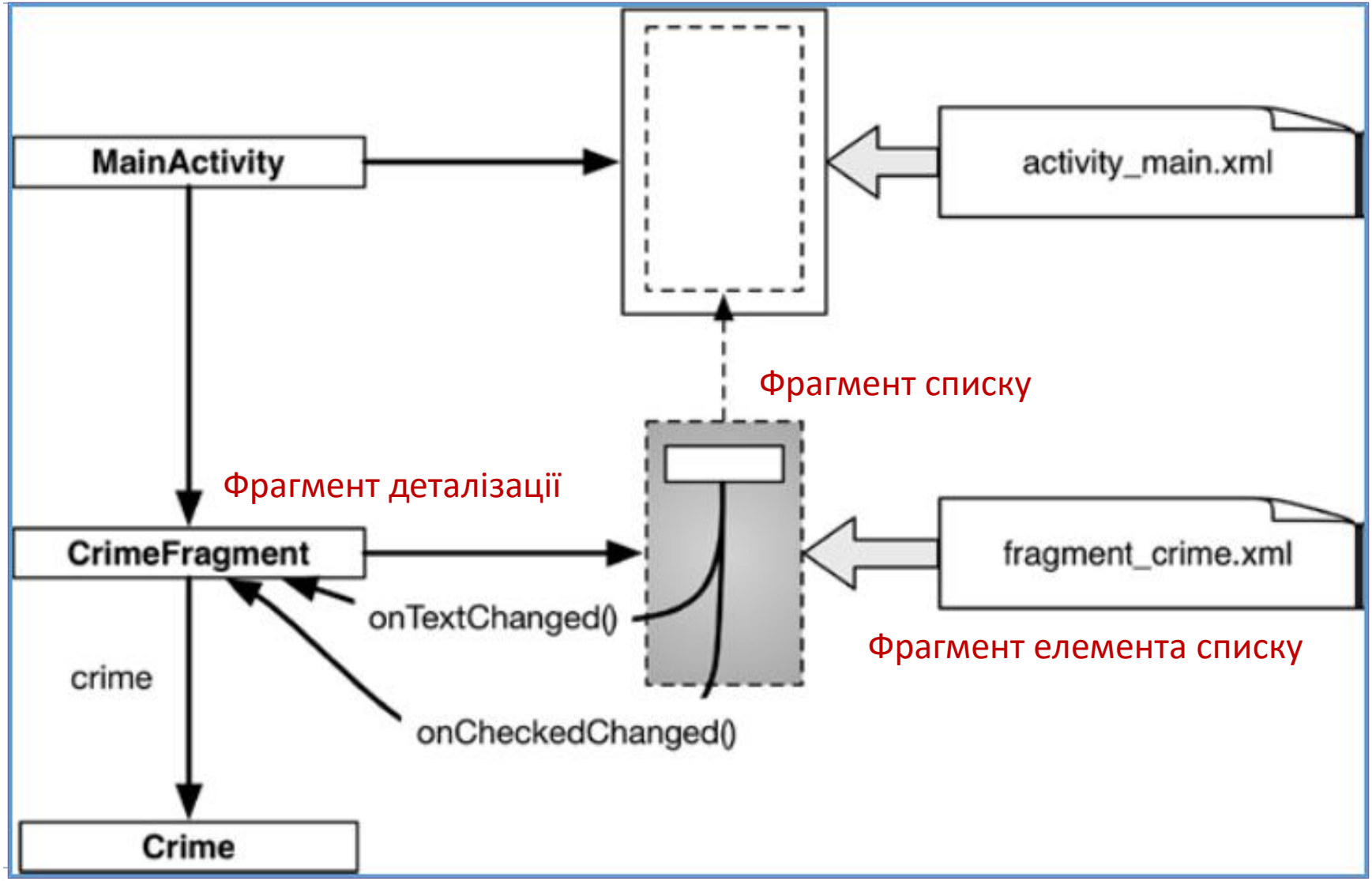
Технологія використання фрагментів

- Забезпечити гнучкість при побудуванні інтерфейсу можливо передавши управління інтерфейсом програми від активності одному або декільком *фрагментам* (*Fragment*). *Фрагмент* є об'єктом контролера, якому активність може довірити виконання операцій. Найчастіше такою операцією є керування інтерфейсом користувача – цілим екраном або його частиною.
- Фрагмент, що управляє інтерфейсом користувача, називається *UI-фрагментом* (*UI-fragment*). UI-фрагмент має власне представлення, яке заповнюється виходячи з файлу макета фрагмента. Представлення активності містить місце, в яке вставляється представлення фрагмента (активність може мати декілька місць для представлення кількох фрагментів). Фрагменти, пов'язані з активністю, можуть використовуватися для формування та зміни екрана відповідно до потреб програми та користувачів. При цьому представлення активності формально залишається незмінним протягом життєвого циклу.

Використання фрагментів у застосунку "список-деталізація"

- У застосунку "список/деталізація" представлення активності будується з фрагмента списку та фрагмента деталізації. Представлення деталізації містить докладну інформацію про вибраний елемент списку. При виборі іншого елемента у списку на екрані з'являється нове деталізоване представлення. Ця зміна представлень відбувається без знищення активності.
- Застосування UI-фрагментів дозволяє розділити інтерфейс застосунку на структурні блоки, а це корисно не тільки для застосунків "список/деталізація". Робота з окремими блоками полегшує розробку інтерфейсів з вкладками, анімованих бічних панелей та багато іншого. Крім того, деякі з нових Android Jetpack API краще працюють саме з фрагментами.

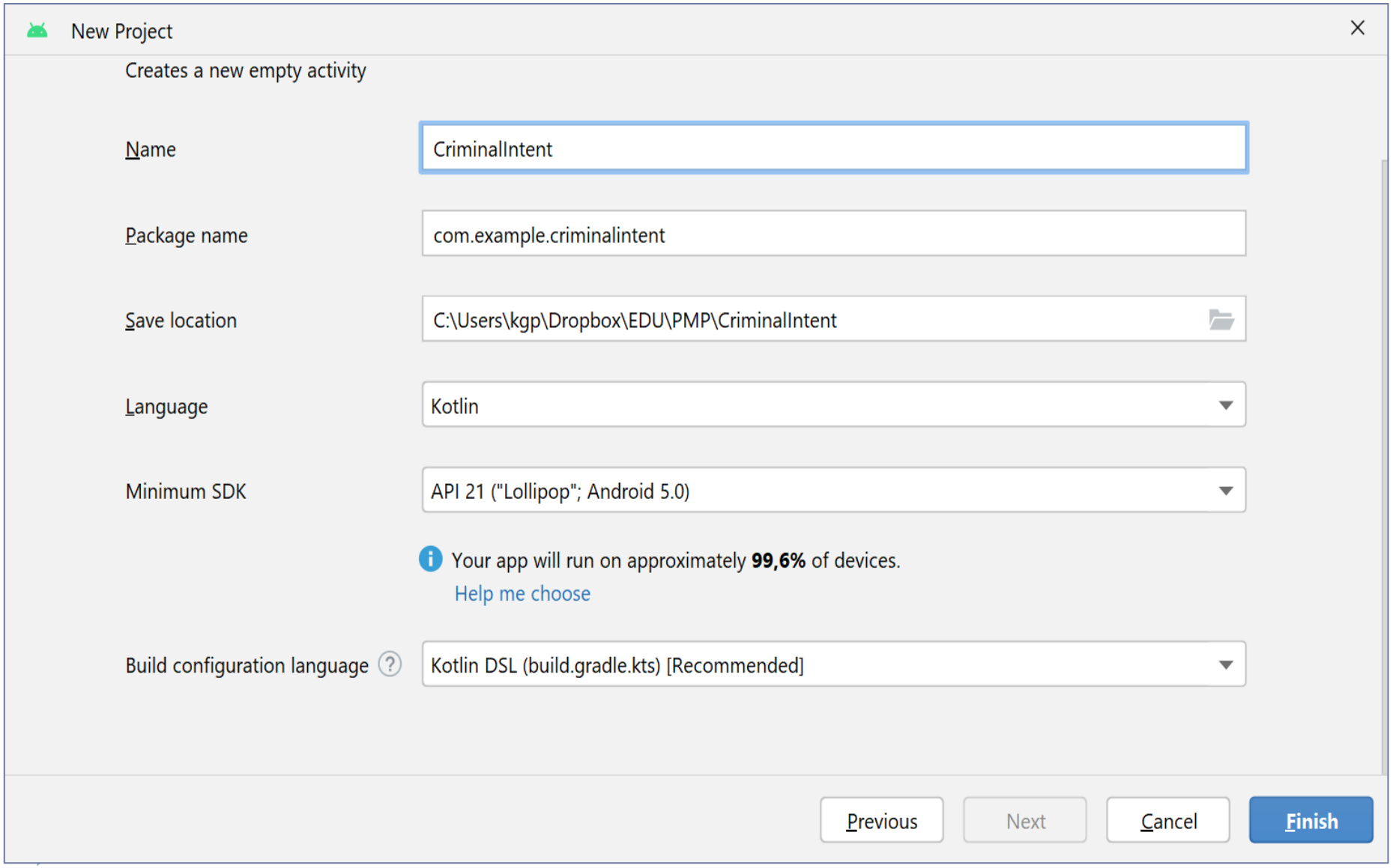
Гнучкість інтерфейсу користувача



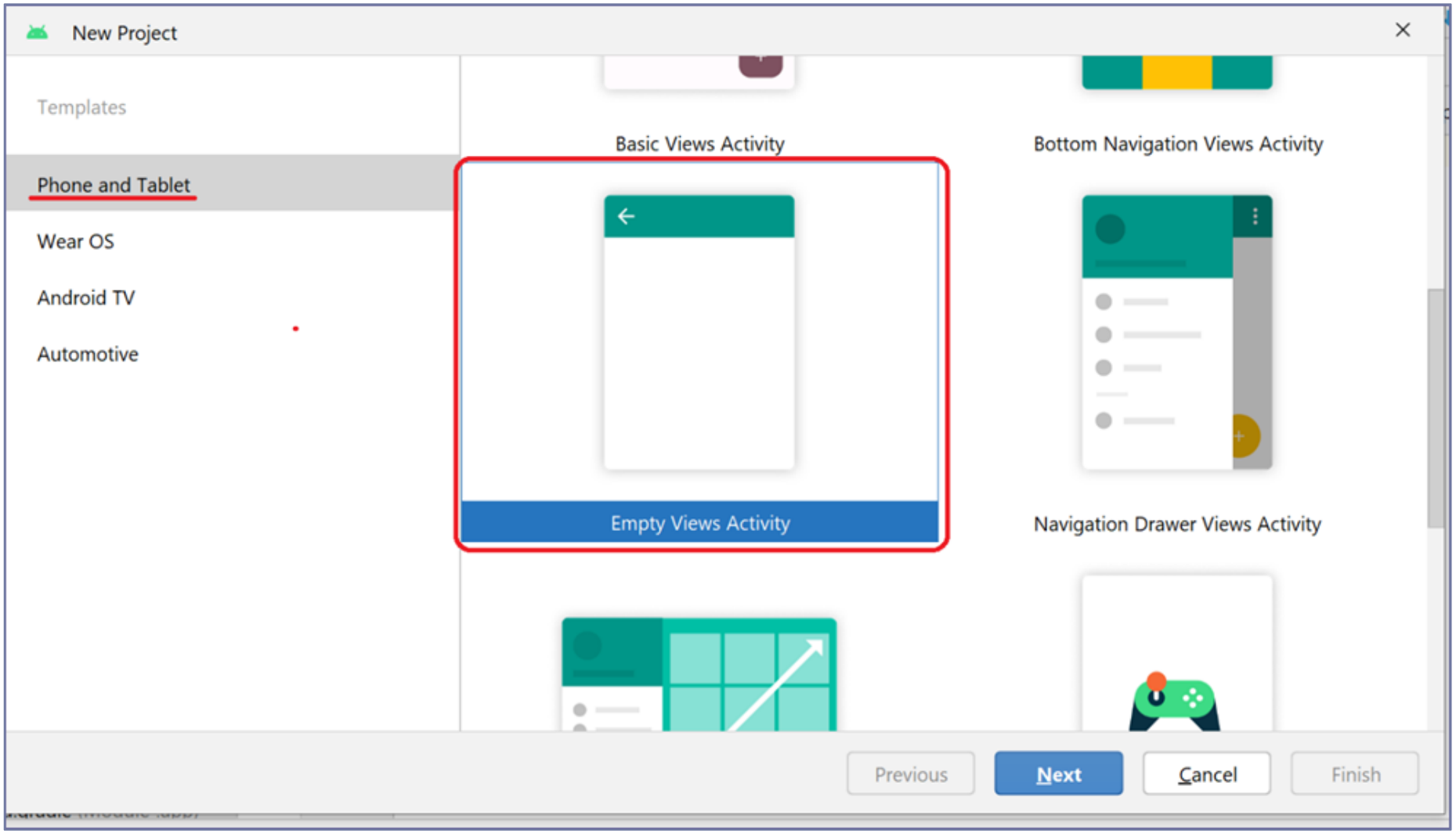
Технологія використання фрагментів

- Фрагменти були введені в API рівня 11 (Android 3.0 Honeycomb – лютий 2011 р) разом з першими планшетами Android. Реалізація фреймворку фрагментів вже була вбудована в пристрої, що працюють на рівні API 11 і вище. Незабаром після цього до бібліотеки підтримки v4 було додано реалізацію класу Fragment для включення підтримки фрагментів на старих пристроях.
- Починаючи з Android 9.0 Pie (API 28 – січень 2022 р) фреймворк-версія фрагментів застаріла. Жодних подальших оновлень цієї версії проводитися не буде. Засоби роботи з фрагментами були перенесені до Jetpack-бібліотеки. Усі подальші оновлення відносяться до Jetpack, а не до фреймворку або фрагментів підтримки v4. Тому у нових проектах необхідно використовувати фрагменти Jetpack, успадковуючи класи користувацьких фрагментів від класу androidx.fragment.app.Fragment.
-

Створення проєкту CriminalIntent



Створення проєкту CriminalIntent

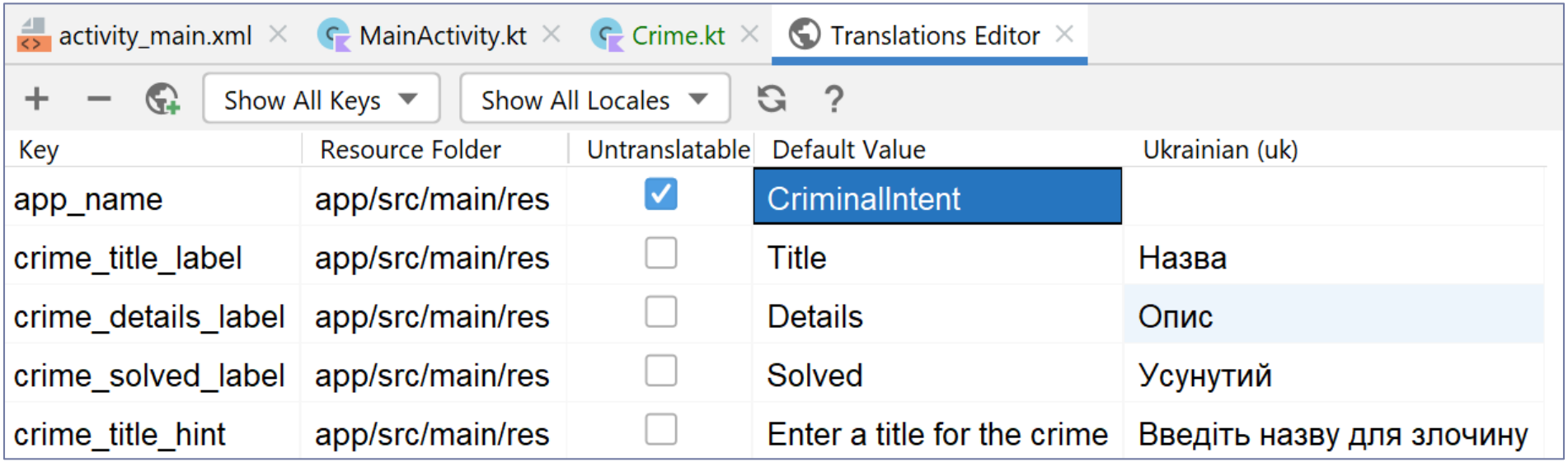


Створення класу моделі

```
package com.example.criminalintent

import java.util.Date
import java.util.UUID

data class Crime(
    val id: UUID = UUID.randomUUID(),
    var title: String = "",
    var date: Date = Date(),
    var isSolved: Boolean = false
);
```

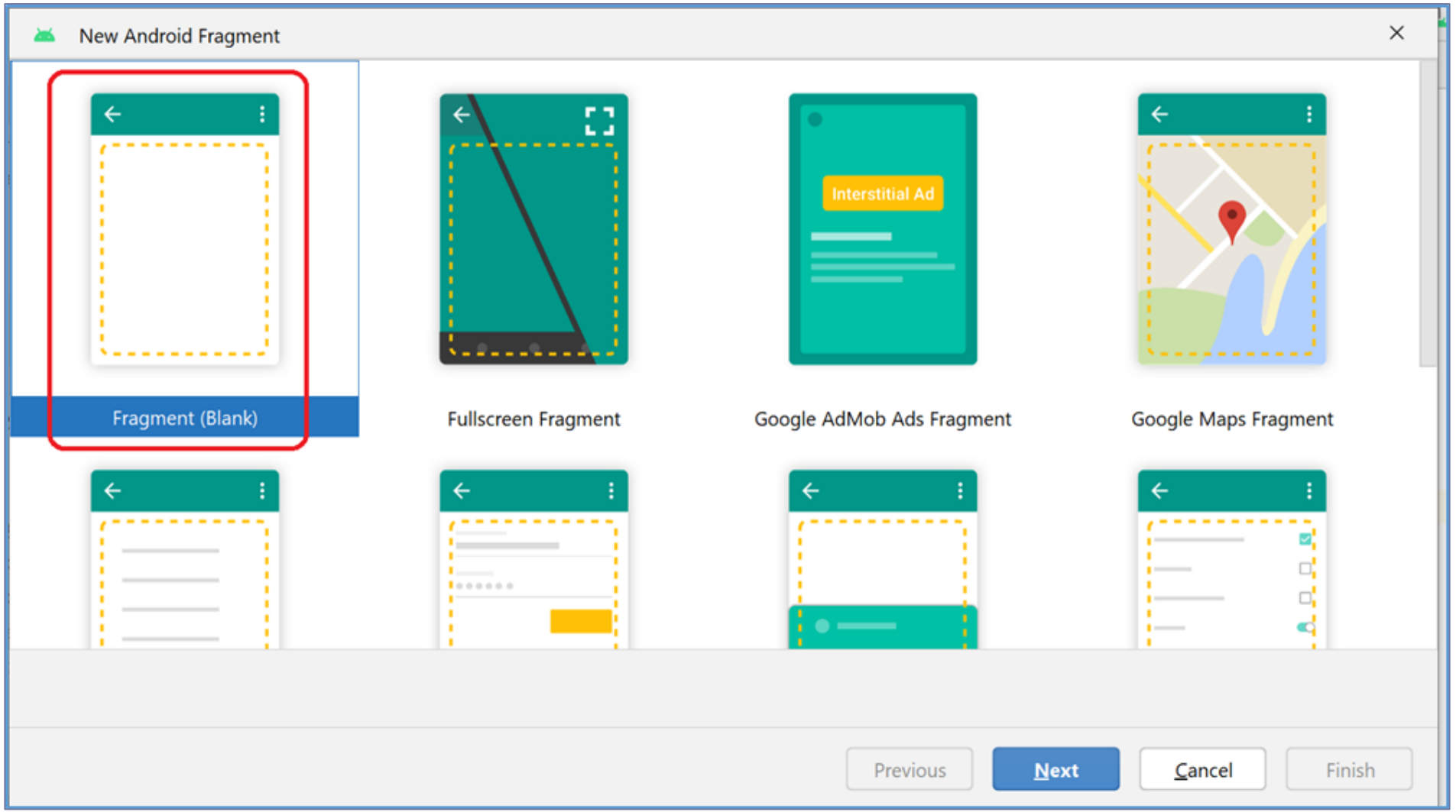
- UUID – це допоміжний клас Java, що входить до інфраструктури Android, він надає простий спосіб генерування універсально-унікальних ідентифікаторів. Ініціалізація змінної date за допомогою конструктора Date за замовчуванням задає поточну дату. Це буде дата злочину за замовчуванням.

Додання рядкових ресурсів

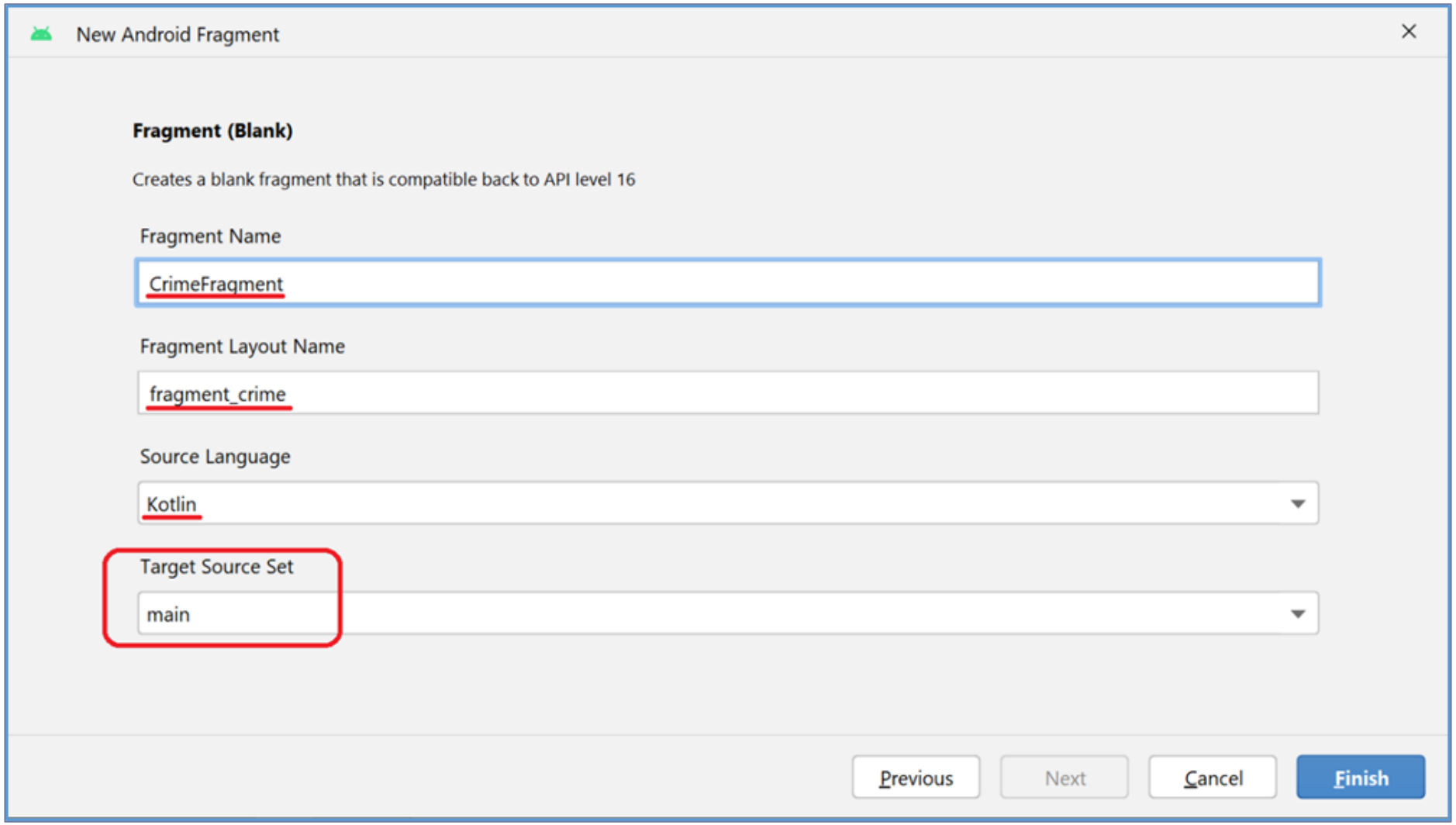


Додання фрагменту деталізації CrimeFragment

New–Fragment/Gallery...



Додання фрагменту деталізації CrimeFragment



Макет фрагменту деталізації CrimeFragment

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="16dp"
    android:orientation="vertical"
    tools:context=".CrimeFragment">
    <TextView
        style="?android:listSeparatorTextViewStyle"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/crime_title_label" />
    <EditText
        android:id="@+id/crime_title"
        android:inputType="text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:paddingTop="12dp"
        android:paddingBottom="12dp"
        android:hint="@string/crime_title_hint" />   ...
```

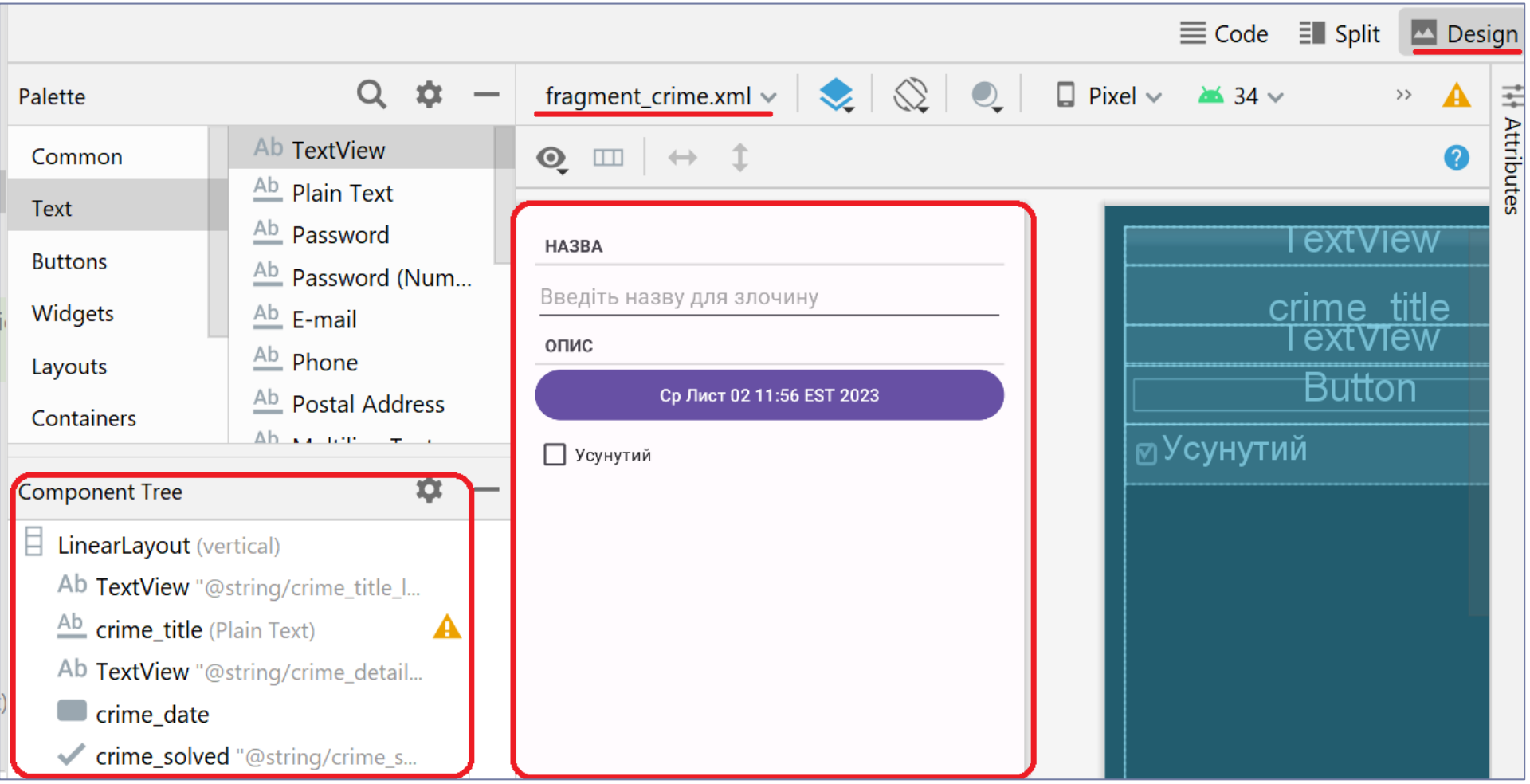
додає лінії-роздільники перед та після віджету

Макет фрагменту деталізації CrimeFragment

додає лінії-роздільники перед та після віджету

```
...
    <TextView
        style="?android:listSeparatorTextViewStyle"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/crime_details_label" />

    <Button
        android:id="@+id/crime_date"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        tools:text="Ср Лист 02 11:56 EST 2023" />

    <CheckBox
        android:id="@+id/crime_solved"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/crime_solved_label" />

</LinearLayout>
```

Макет фрагменту деталізації CrimeFragment



Клас фрагменту деталізації CrimeFragment

```
package com.example.criminalintent

import android.*
import androidx.fragment.app.Fragment

private const val TAG = "CrimeFragment"

class CrimeFragment : Fragment() {

    private lateinit var crime: Crime
    private lateinit var titleField: EditText       //Crime title field
    private lateinit var dateButton: Button         //Crime date
    private lateinit var solvedCheckBox: CheckBox   //Is the crime solved

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        Log.d(TAG, "onCreate(Bundle?) called")
        crime = Crime()
    }

...
```

Клас фрагменту деталізації CrimeFragment

```
...
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        Log.d(TAG, "onCreateView(...) called")
        val view = inflater.inflate(R.layout.fragment_crime, container, false)
        titleField = view.findViewById(R.id.crime_title) as EditText
        dateButton = view.findViewById(R.id.crime_date) as Button
        solvedCheckBox = view.findViewById(R.id.crime_solved) as CheckBox

        dateButton.apply {
            text = crime.date.toString()
            isEnabled = false       //Button blocked
        }

        return view
    }
...
```

функція життєвого циклу фрагменту

Виклик ідентифікаторів ресурсів з View фрагмента, а не з активності

Клас фрагменту деталізації CrimeFragment

функція життєвого циклу фрагменту

```
...
override fun onStart() {
    super.onStart()
    Log.d(TAG, "onStart() called")
    /*Anonymous implementation TextWatcher Listener interface*/
    val titleWatcher = object : TextWatcher {
        override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int,
                                       after: Int) {
            TODO("Not yet implemented")
        }
        override fun onTextChanged(s: CharSequence?, start: Int, before: Int,
                                   count: Int) {
            crime.title = s.toString()
        }
        override fun afterTextChanged(s: Editable?) {
            TODO("Not yet implemented")
        }
    }
    titleField.addTextChangedListener(titleWatcher)
...
```

Клас фрагменту деталізації CrimeFragment

```
...
        solvedCheckBox.apply {
            /*In Kotlin, the underscore character ( _ ) is used to denote unused parameters
              in a function, lambda, or destructuring declaration.
              The abstract method of CompoundButton's inner interface
              public static interface OnCheckedChangeListener:
              void onCheckedChanged(CompoundButton buttonView, boolean isChecked);
              has parameters:
                buttonView – The compound button view whose state has changed.
                isChecked – The new checked state of buttonView..*/
            setOnCheckedChangeListener { _, isChecked ->
                crime.isSolved = isChecked
            }
        }
    }
}
```

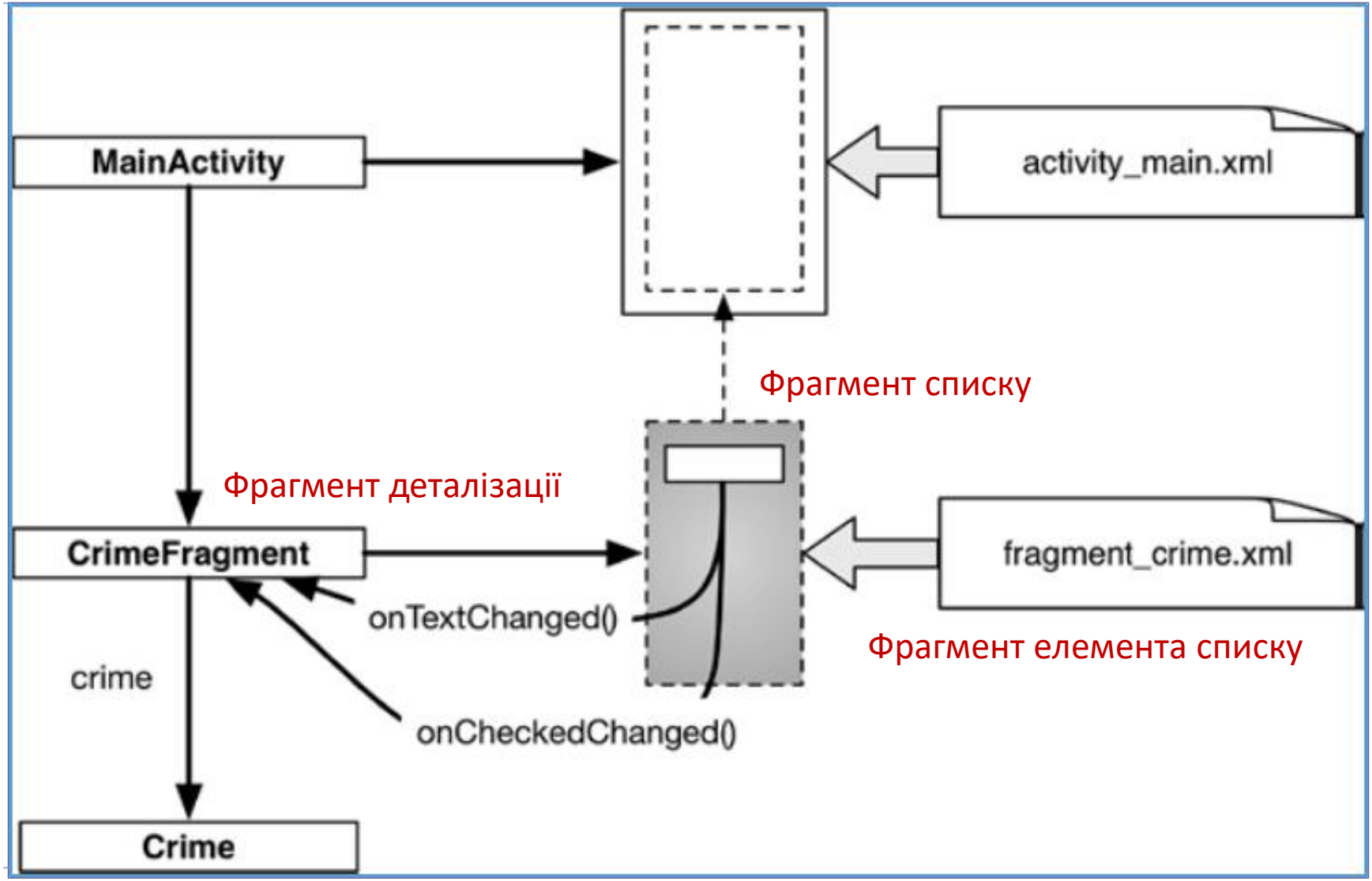
- Слухач `TextWatcher` налаштовується у функції `onStart()`, оскільки він спрацьовує не тільки при взаємодії з користувачем, але і при відновленні стану віджету, наприклад, при повороті. Натомість слухачі, які реагують лише на взаємодію з користувачем, такі як `OnClickListener`, не гублять свій стан при змінах конфігурації, тому їх можна налаштувати в методі `onCreate(...)` перед будь-яким відновленням стану.

Макет активності-хостингу UI-фрагментів activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"/>
```

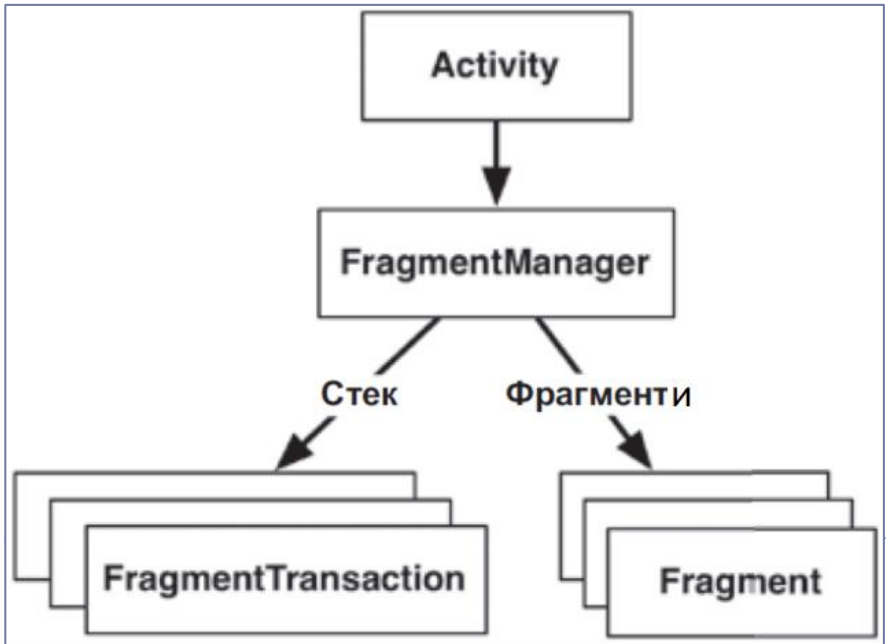
- Елемент FrameLayout стане контейнерним представленням для фрагменту списку злочинів. Зверніть увагу, що це представлення є абсолютно універсальним: воно не прив'язується ні до якого фрагменту. Ми можемо використовувати один макет для хостингу різних фрагментів.
- Хоча макет activity_crime.xml складається виключно з контейнерного представлення одного фрагмента, він може бути складнішим: визначати кілька контейнерних представлень та власні віджети.

Макет активності-хостингу UI-фрагментів activity_main.xml



FragmentManager

- Коли в Android 3.0 (Honeycomb) з'явився клас Fragment, у клас Activity були внесені зміни: до нього був доданий компонент, званий FragmentManager. FragmentManager працює зі списком фрагментів та зворотним стеком транзакцій (який буде розглянутий далі).
- FragmentManager відповідає за додавання представлень фрагментів до ієрархії представлень активності та за управління життєвими циклами фрагментів.



Клас активності-хостингу UI-фрагментів MainActivity

```
package com.example.criminalintent

import androidx.appcompat.app.AppCompatActivity
import android.*

private const val TAG = "MainActivity"

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        Log.d(TAG, "onCreate(Bundle?) called")
        setContentView(R.layout.activity_main)

        val currentFragment =
                supportFragmentManager.findFragmentById(R.id.fragment_container)

        if(currentFragment == null){
            val fragment = CrimeFragment()    //temporal - instead CrimeListFragment
            supportFragmentManager
                .beginTransaction()
                .add(R.id.fragment_container, fragment)
                .commit()
        }
} }
```

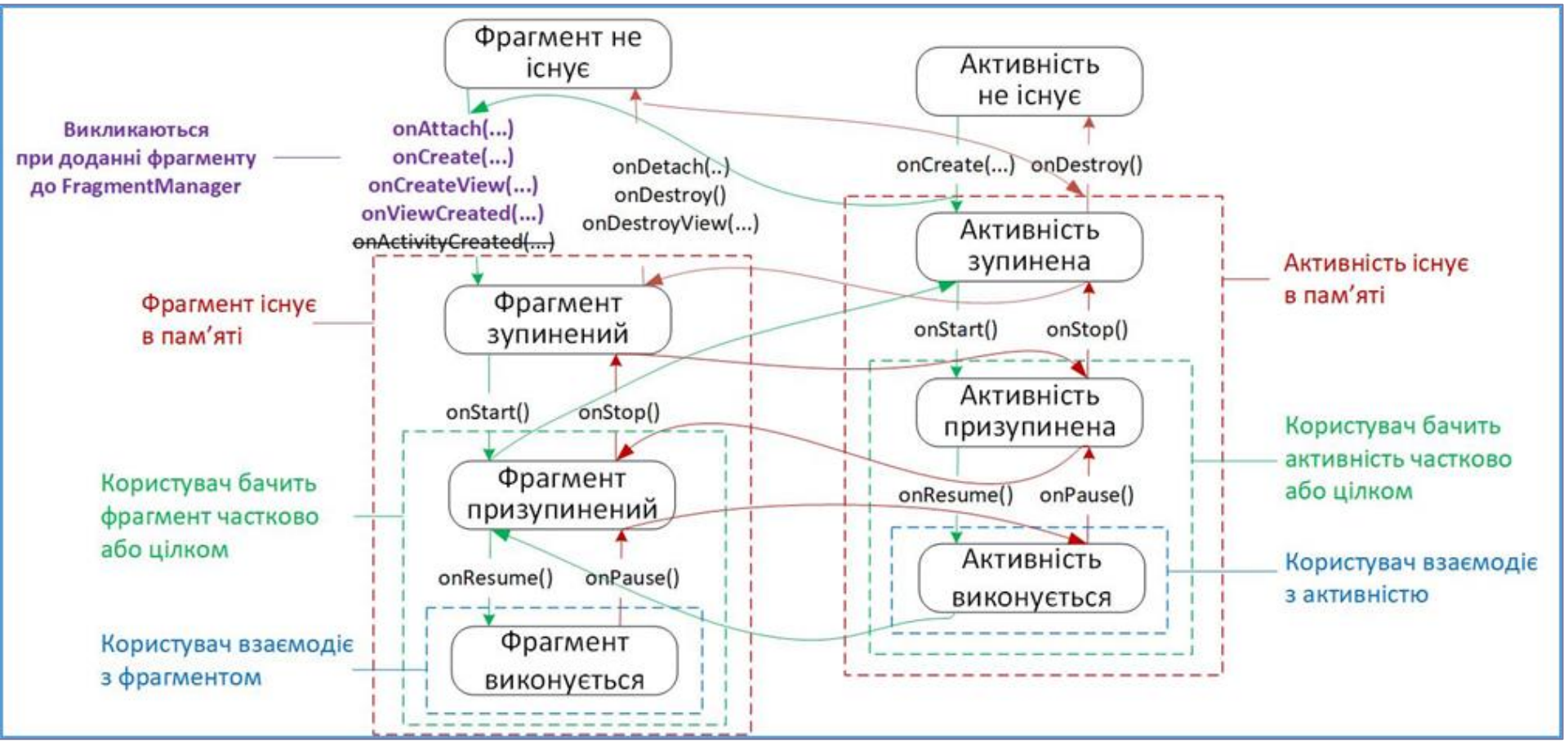
Властивість класу FragmentActivity

Додання фрагменту у транзакції (можуть виконуватися видалення, під'єднання, від'єднання та заміна фрагментів)

Використання транзакцій дозволяє виконувати групу операцій, наприклад, додавати декілька фрагментів до різних контейнерів під час виконання програми

Активність-хостинг з фрагментом

Життєвий цикл фрагментів

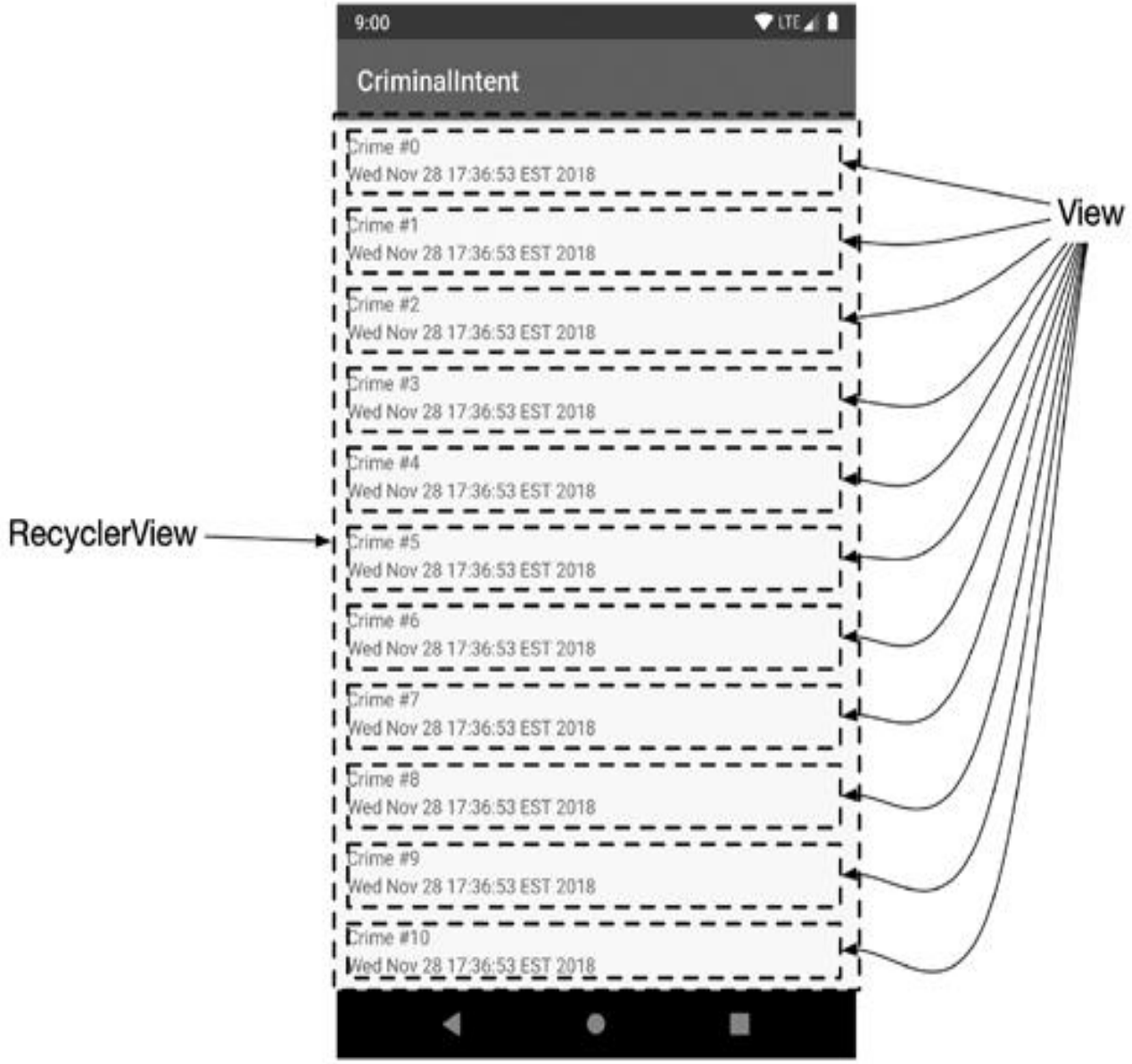


Оскільки фрагмент працює від імені активності, його стан має бути узгодженим зі станом активності. На відміну від активності, функції життєвого циклу фрагмента викликаються не операційною системою, а FragmentManager, який виконує ці функції. Операційна система нічого не знає про фрагменти, які вона використовує для керування. Фрагменти – це внутрішня кухня активності.

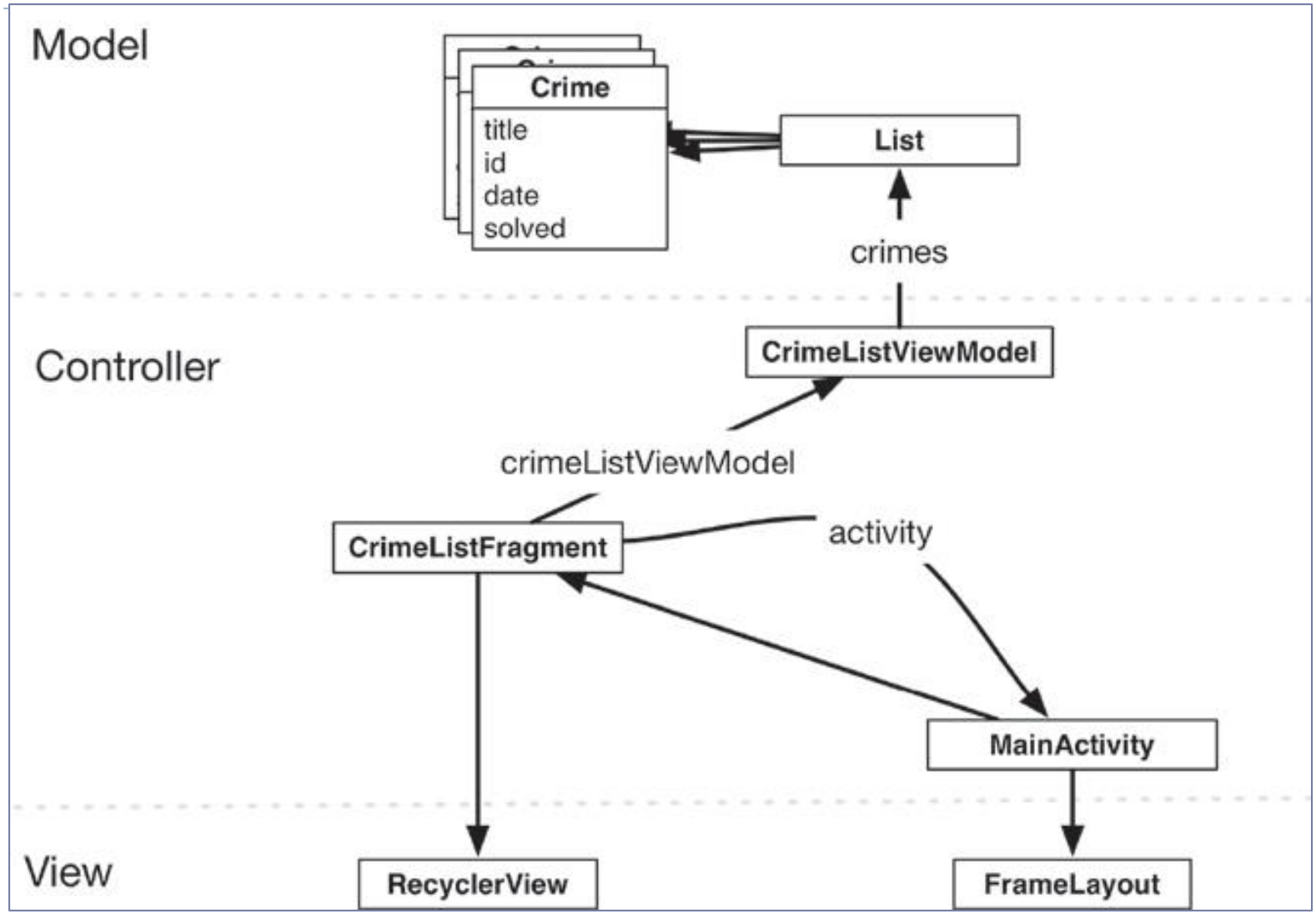
Життєвий цикл фрагментів

- Якщо фрагмент додається у той час, коли активність вже знаходиться в стані зупинки, призупинки або виконання, FragmentManager негайно проводить фрагмент через всі етапи, необхідні для узгодження його стану з відповідним станом активності.
- Наприклад, при додаванні фрагмента до активності, що вже перебуває в стані виконання, фрагмент отримає виклики onAttach(Context), onCreate(Bundle?), onCreateView(...), onViewCreated(…), onActivityCreated(Bundle), onStart() і onResume().
- Після того, як стан фрагмента буде узгоджений зі станом активності, об'єкт FragmentManager хост-активності буде викликати подальші функції життєвого циклу приблизно одночасно з отриманням відповідних викликів від ОС для синхронізації стану фрагмента зі станом активності.

Виведення списків за допомогою RecyclerView



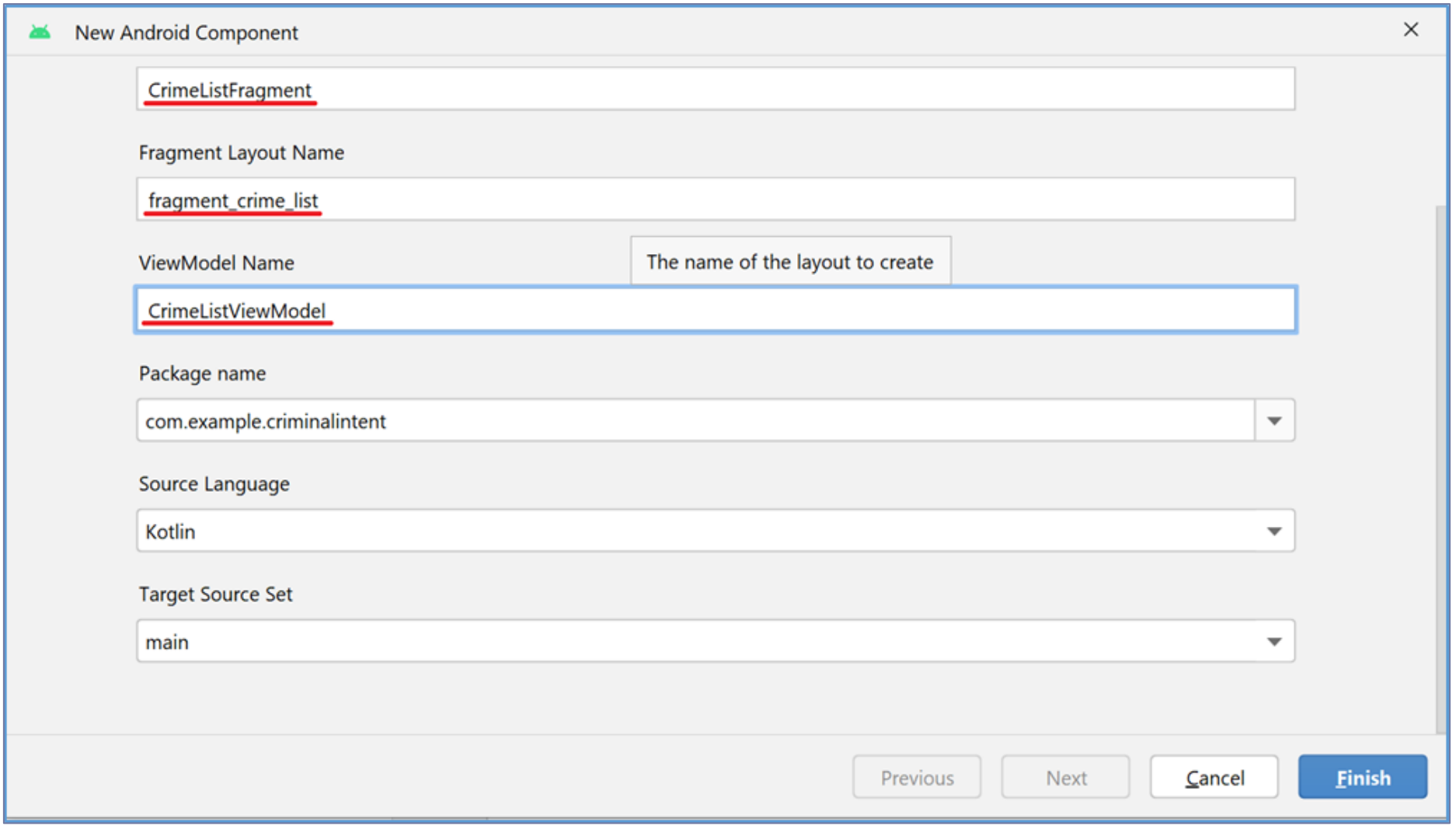
Виведення списків за допомогою RecyclerView



Архітектура застосунку CriminalIntent з підтримкою списку

Додання фрагменту списку CrimeListFragment

File–New–Fragment–Fragment (with ViewModel)



Клас ViewModel списку

```
package com.example.criminalintent
import androidx.lifecycle.ViewModel
class CrimeListViewModel : ViewModel() {
    val crimes = mutableListOf<Crime>()
    init {
        for (i in 0 until 100){
            val crime = Crime()
            crime.title = "Crime #$i"
            crime.isSolved = i % 2 == 0
            crimes += crime
        }
    }
}
```

Генерування злочинів (тимчасовий плейсхолдер)

Клас фрагменту списку CrimeListFragment

```
package com.example.criminalintent

import ...

private const val TAG = "CrimeListFragment"

class CrimeListFragment : Fragment() {

    companion object {
        fun newInstance() = CrimeListFragment()
    }

    private val viewModel: CrimeListViewModel by lazy {
        ViewModelProvider(this).get(CrimeListViewModel::class.java)
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        Log.d(TAG,"Total crimes: ${viewModel.crimes.size}")
        return inflater.inflate(R.layout.fragment_crime_list, container, false)
    }
}
```

Життєвий цикл ViewModel з фрагментами

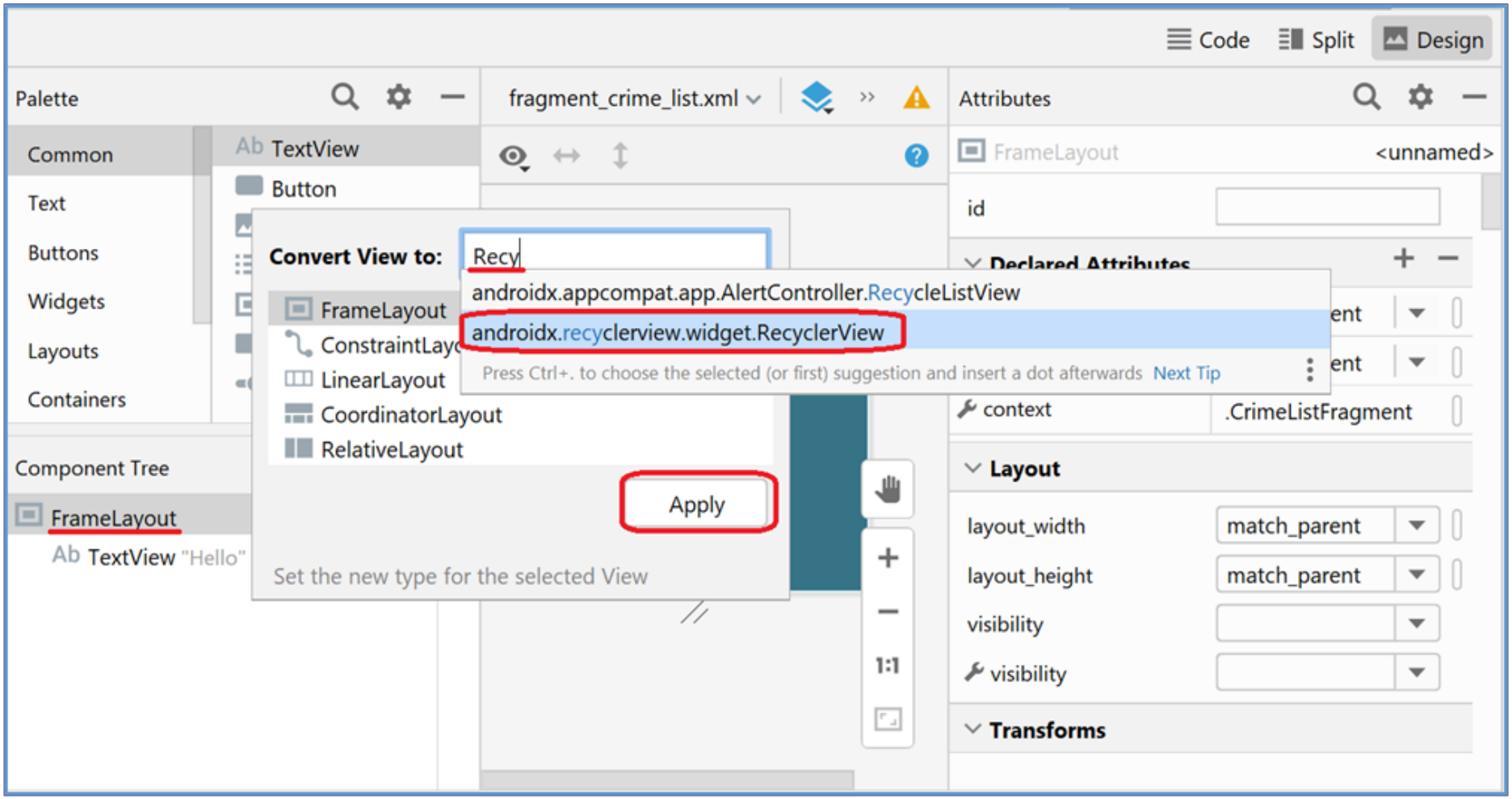
- Об'єкт ViewModel залишатиметься активним, доки віджет фрагмента знаходиться на екрані. ViewModel зберігається при повороті (навіть якщо екземпляр фрагмента не зберігається) і буде доступним для нового екземпляра фрагмента.
- Але об'єкт ViewModel знищується після знищення фрагмента. Це може статися, коли користувач натискає кнопку "Назад", закриваючи екран. Це також може статися, якщо хост-активність замінює фрагмент на інший. Хоча на екрані відображається та ж активність, старий фрагмент, і пов'язаний з ним об'єкт ViewModel будуть знищені, оскільки вони більше не потрібні.
- Є одне виключення з цієї поведінки – коли активність замінює поточний фрагмент іншим, а транзакція повертається до стеку, фрагмент і його ViewModel знищені не будуть. Якщо користувач натискає кнопку "Назад", транзакція фрагмента відновлюється. Оригінальний екземпляр фрагмента поміщається назад на екран, і всі дані ViewModel зберігаються.

Клас активності

```
package com.example.criminalintent

import ...

private const val TAG = "MainActivity"

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        Log.d(TAG, "onCreate(Bundle?) called")
        setContentView(R.layout.activity_main)

        val currentFragment =
                supportFragmentManager.findFragmentById(R.id.fragment_container)

        if(currentFragment == null){
            val fragment = CrimeListFragment.newInstance()
            supportFragmentManager
                .beginTransaction()
                .add(R.id.fragment_container, fragment)
                .commit()
        }
    }
}
```

Відображення списку у RecyclerView

Заміна кореневого елемента fragment_crime_list.xml на androidx.recyclerview.widget.RecyclerView



Макет fragment_crime_list.xml

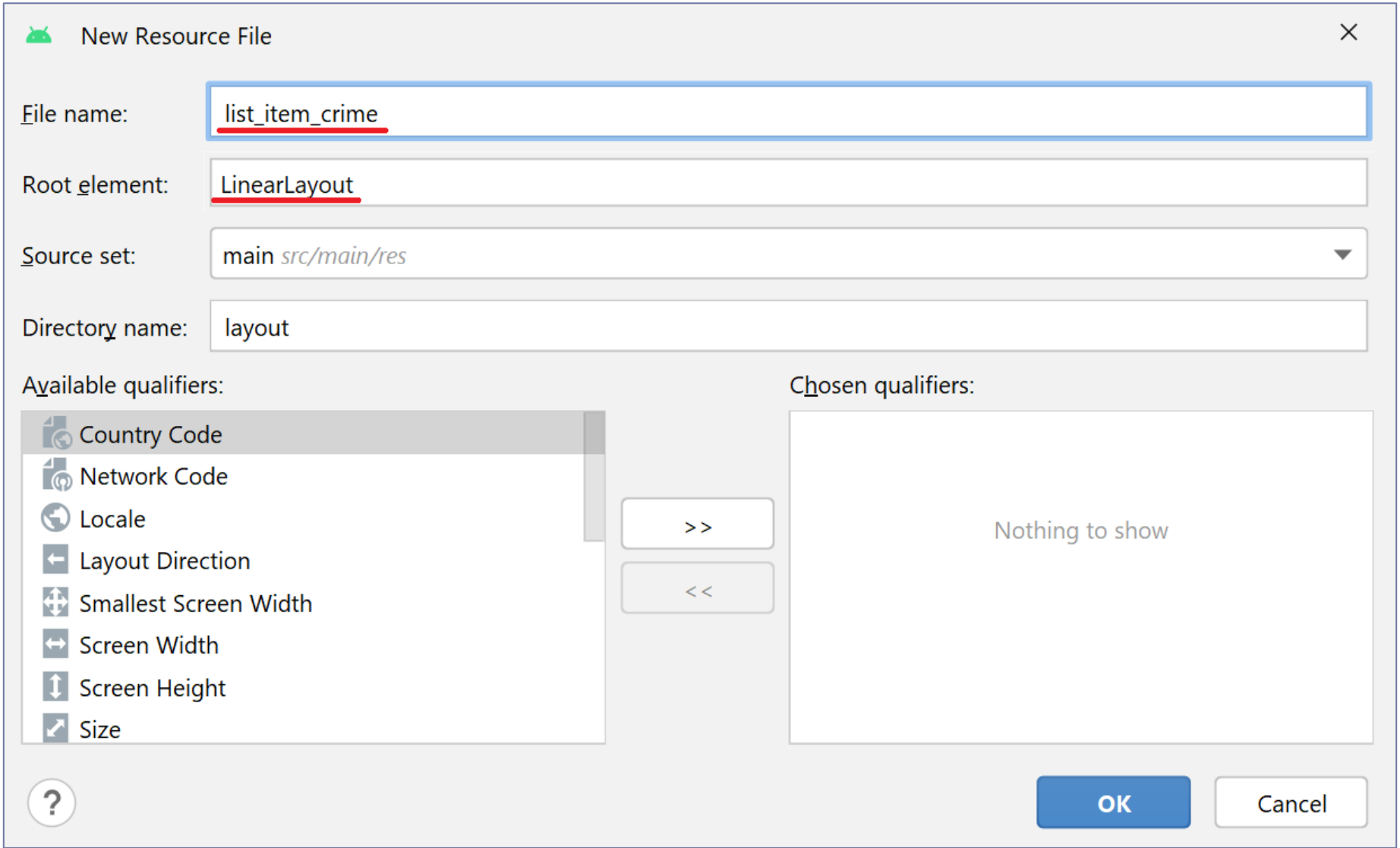
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.recyclerview.widget.RecyclerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/crime_recycler_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".CrimeListFragment"/>
```

Клас фрагменту списку CrimeListFragment - підключення макета

```
package com.example.criminalintent

import ...

private const val TAG = "CrimeListFragment"

class CrimeListFragment : Fragment() {

    private lateinit var crimeRecyclerView: RecyclerView

    companion object {
        fun newInstance() = CrimeListFragment()
    }

    private val viewModel: CrimeListViewModel by lazy { .. }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
    ): View? {
        Log.d(TAG,"Total crimes: ${viewModel.crimes.size}")
        val view = inflater.inflate(R.layout.fragment_crime_list, container, false)
        crimeRecyclerView = view.findViewById(R.id.crime_recycler_view)
                                                    as RecyclerView
        crimeRecyclerView.layoutManager = LinearLayoutManager(context)
        return view
    } }
```

Макет елемента списку list_item_crime.xml

File–New–Layout Resource File



Макет елемента списку list_item_crime.xml

File–New–Layout Resource File

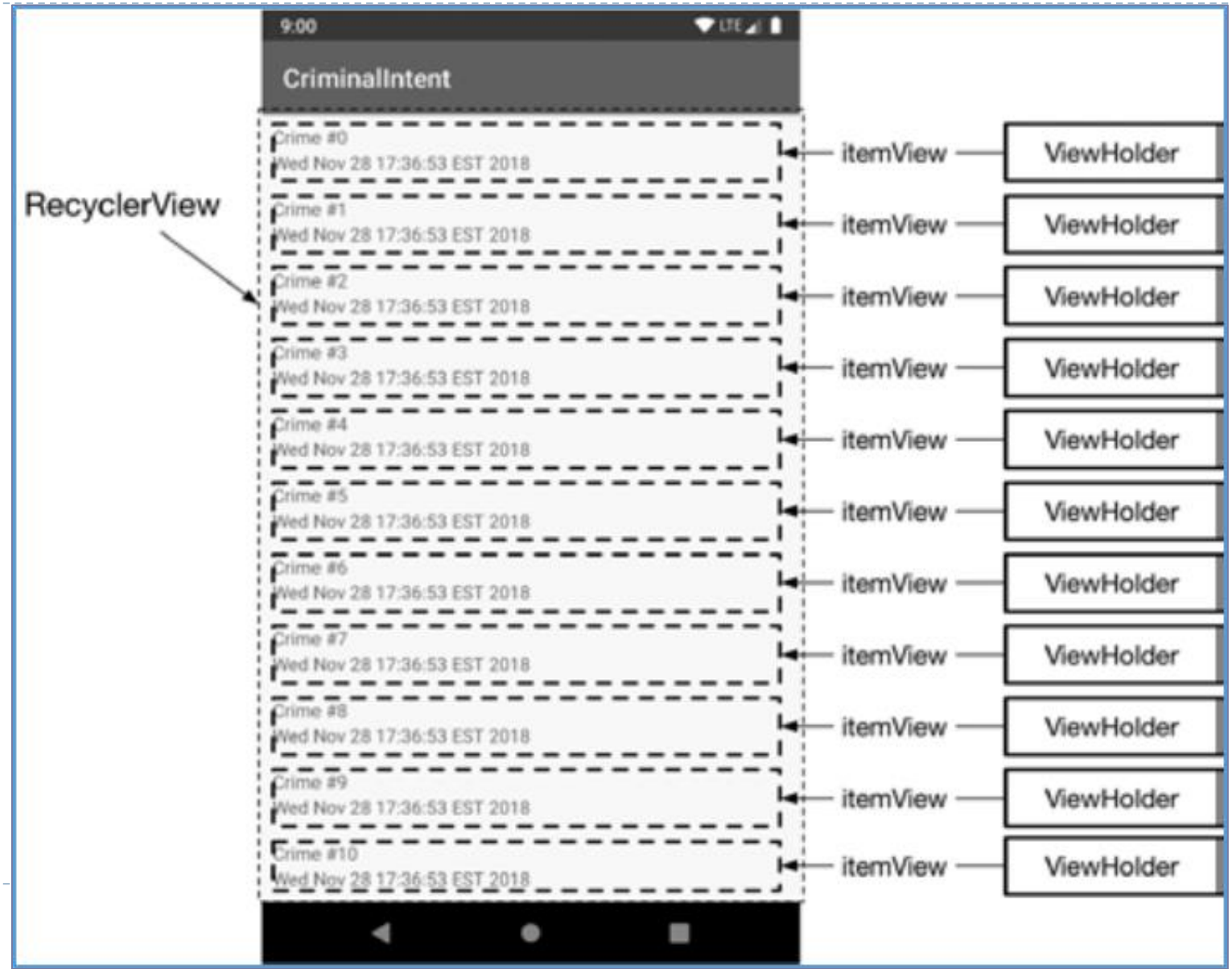
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:padding="8dp">

    <TextView
        android:id="@+id/list_item_crime_title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/crime_title_label" />

    <TextView
        android:id="@+id/list_item_crime_date"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/crime_date" />
</LinearLayout>
```

доданий рядковий ресурс
(зі значеннями *Date* та *Дата*)

RecylerView.ViewHolder

- Об'єкт RecyclerView очікує, що віджети, які він повинен виводити для кожного елемента списку будуть надаватися об'єктом RecyclerView.ViewHolder.
- Конструктор CrimeHolder приймає посилання на представлення, яке передається до конструктора RecyclerView.ViewHolder. Базовий клас ViewHolder має властивість itemView, за допомогою якої можна отримувати посилання на віджети макету елементів списку.
- RecyclerView ніколи не створює об'єкти View особисто, він завжди створює ViewHolder, які виводять свої itemView.

RecylerView.ViewHolder



Клас фрагменту списку CrimeListFragment - додання ViewHolder-а

```
package com.example.criminalintent
import ...
private const val TAG = "CrimeListFragment"
class CrimeListFragment : Fragment() {
    ...
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
    ): View? {
        ...
    }
    private inner class CrimeHolder(view: View) : RecyclerView.ViewHolder(view) {
        val titleTextView = itemView.findViewById(R.id.list_item_crime_title)
                                                    as TextView
        val dateTextView = itemView.findViewById(R.id.list_item_crime_date)
                                                    as TextView
    }
}
```

RecyclerView.Adapter<CrimeHolder>

- Насправді об'єкт RecyclerView не створює ViewHolder особисто. Натомість використовується адаптер, який є об'єктом контролера і знаходиться між RecyclerView і наборами даних, які відображає RecyclerView.
- Об'єкт адаптера виконує такі функції:
  - створення за запитом необхідних об'єктів ViewHolder;
  - зв'язування об'єктів ViewHolder з даними шару моделі.
- Об'єкт RecyclerView виконує такі функції:
  - надсилає адаптеру запит на створення нового об'єкта ViewHolder;
  - надсилає адаптеру запит на зв'язування об'єкта ViewHolder з елементом даних, який відповідає поточній позиції.

Клас фрагменту списку CrimeListFragment - додання Adapter-а

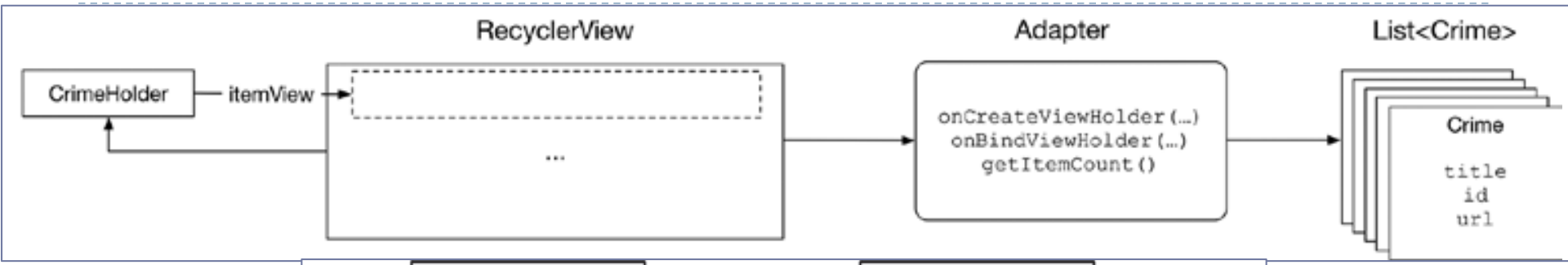
```
package com.example.criminalintent
import ...
private const val TAG = "CrimeListFragment"
class CrimeListFragment : Fragment() {
    private lateinit var crimeRecyclerView: RecyclerView
    ...
    private inner class CrimeHolder(view: View) : RecyclerView.ViewHolder(view) {...}
    private inner class CrimeAdapter(var crimes: List<Crime>) :
                                                    RecyclerView.Adapter<CrimeHolder>()
{
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
                                                    CrimeHolder {
        val view = layoutInflater.inflate(R.layout.list_item_crime, parent, false)
        return CrimeHolder(view)
    }
...
```

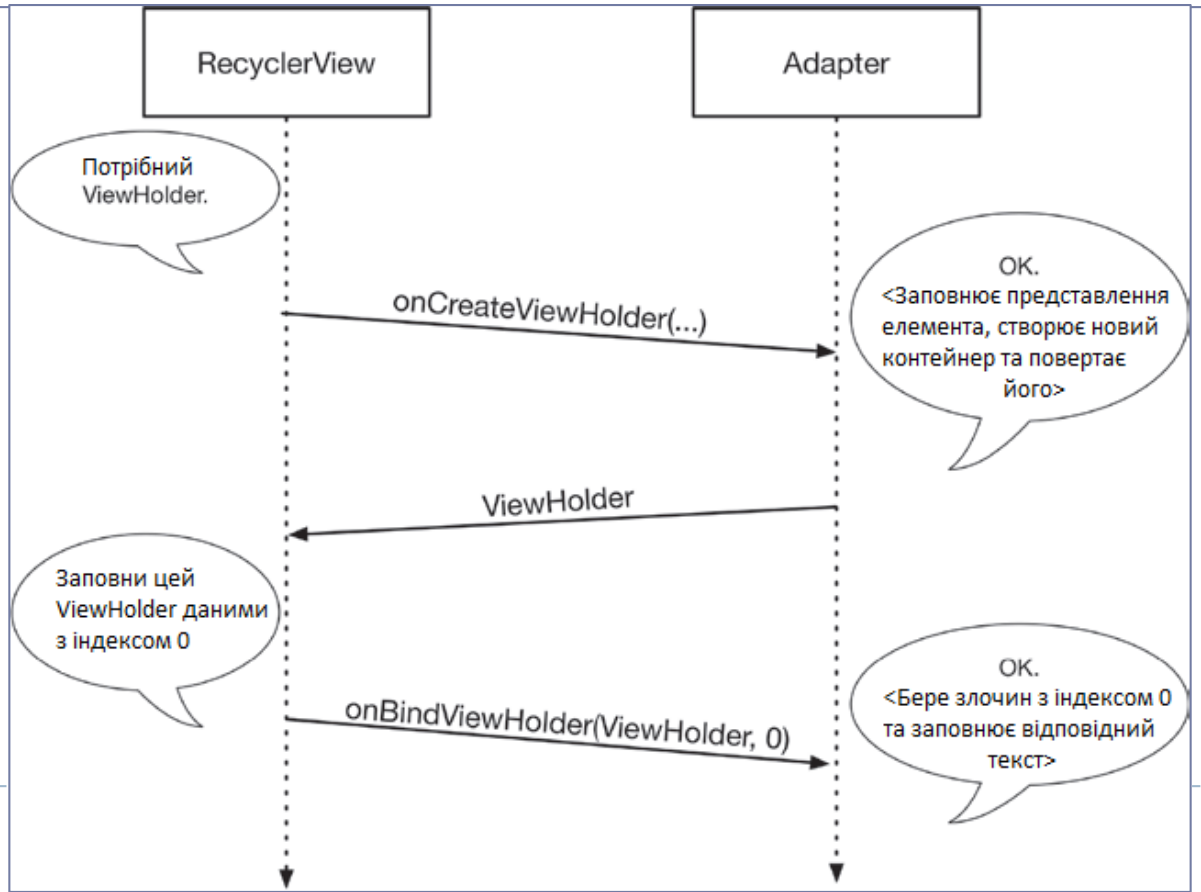
Клас фрагменту списку CrimeListFragment - додання Adapter-а

```
...
    override fun onBindViewHolder(holder: CrimeHolder, position: Int) {
        val crime = crimes[position]
        holder.apply {
            titleTextView.text = crime.title
            dateTextView.text = crime.date.toString()
        }
    }
    override fun getItemCount() = crimes.size
}
}
```

- Функція `Adapter.onCreateViewHolder(...)` відповідає за створення представлення з макетом list_item_crime.xml, обертає його у об'єкт CrimeHolder, який повертає. (На даний момент Ви можете ігнорувати параметри `onCreateViewHolder(...)`.
- Функція `Adapter.onBindViewHolder(holder: CrimeHolder, position: Int)` відповідає за заповнення даного холдера даними злочину з відповідним значенням індексу `position` у списку злочинів.
- Функція `Adapter.getItemCount()` повертає кількість елементів у списку злочинів, відповідаючи на запит об'єкта RecyclerView.

Робота адаптера





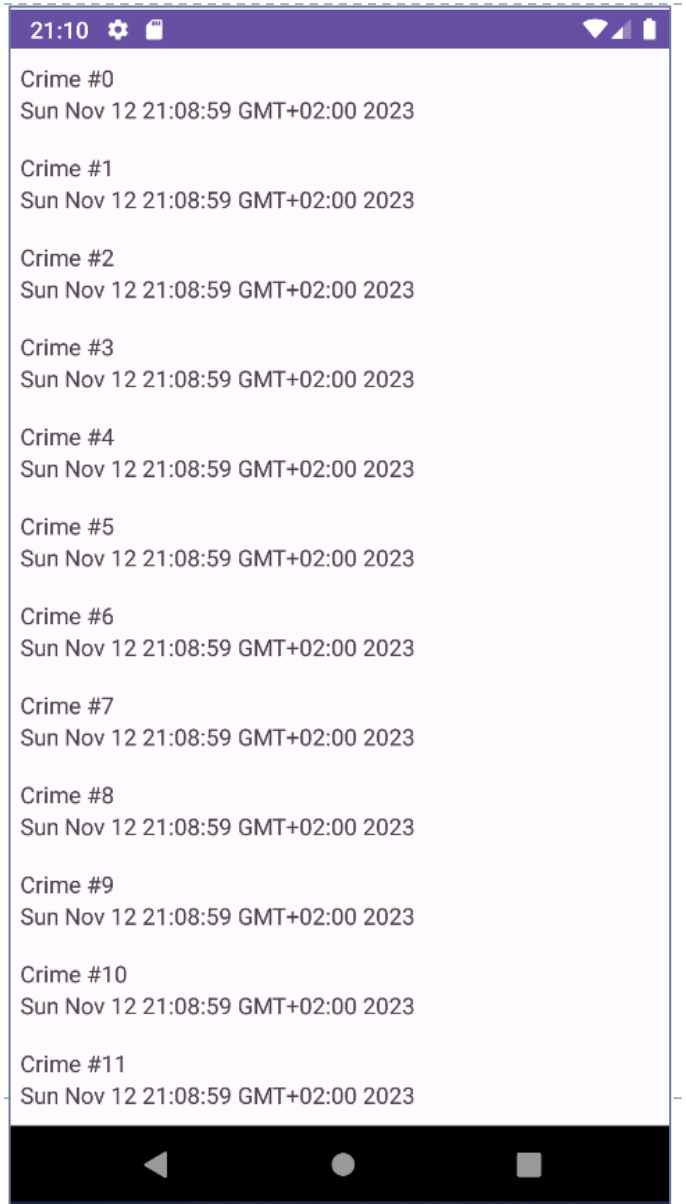
Підключення CrimeAdapter до RecyclerView

```
package com.example.criminalintent

import ...

private const val TAG = "CrimeListFragment"

class CrimeListFragment : Fragment() {

    private lateinit var crimeRecyclerView: RecyclerView
    private var adapter: CrimeAdapter? = null
...
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        Log.d(TAG, "Total crimes: ${crimeListViewModel.crimes.size}")
        val view = inflater.inflate(R.layout.fragment_crime_list, container, false)
        crimeRecyclerView = view.findViewById(R.id.crime_recycler_view)
                                                    as RecyclerView
        crimeRecyclerView.layoutManager = LinearLayoutManager(context)
        updateUI()
        return view
    }
...
```

Підключення CrimeAdapter до RecyclerView

```
...
    private fun updateUI() {
        val crimes = crimeListViewModel.crimes
        adapter = CrimeAdapter(crimes)
        crimeRecyclerView.adapter = adapter
    }

    private inner class CrimeHolder(view: View) : RecyclerView.ViewHolder(view) {...}

    private inner class CrimeAdapter(var crimes: List<Crime>) :
                                                    RecyclerView.Adapter<CrimeHolder>() {

        ...
    }
}
```

Підключення CrimeAdapter до RecyclerView



RecyclerView

- Замість створення 100 об'єктів View, RecyclerView створює їх стільки, скільки потрібно для заповнення екрана. Коли елемент зникає з екрана, RecyclerView використовує його представлення для інших елементів. Коли створено достатньо об'єктів ViewHolder для заповненя екрану, RecyclerView перестає викликати на onCreateViewHolder(..). Натомість він заощаджує час і пам'ять шляхом повторного використання старих об'єктів ViewHolder і передає їх до onBindViewHolder(ViewHolder, Int).
- В даний момент Adapter прив'язує дані злочину до текстових віджетів у функції Adapter.onBindViewHolder(...). Але краще розділити завдання між холдером та адаптером: адаптер повинен знати якомога менше про внутрішню кухню і дані холдера. Тому перенесемо код, який виконує прив'язку даних злочину до текстових віджетів до CrimeHolder.

Перенесення прив'язки даних до віджетів з CrimeAdapter до CrimeHolder

```
package com.example.criminalintent

import ...

private const val TAG = "CrimeListFragment"

class CrimeListFragment : Fragment() {

    private lateinit var crimeRecyclerView: RecyclerView
    private var adapter: CrimeAdapter? = null

    companion object {
        fun newInstance() = CrimeListFragment()
    }

    private val crimeListViewModel: CrimeListViewModel by lazy {
        ViewModelProvider(this).get(CrimeListViewModel::class.java)
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
    ): View? {...}

    private fun updateUI() { ... }

...
```

Перенесення прив'язки даних до віджетів з CrimeAdapter до CrimeHolder

```
...
private inner class CrimeHolder(view: View) : RecyclerView.ViewHolder(view) {
    private lateinit var crime: Crime
    private val titleTextView = itemView.findViewById(R.id.list_item_crime_title)
                                                        as TextView
    private val dateTextView = itemView.findViewById(R.id.list_item_crime_date)
                                                        as TextView
    fun bind(crime: Crime) {
        this.crime = crime
        titleTextView.text = this.crime.title
        val dateFormat = DateFormat.getLongDateFormat(context)
        val timeFormat = DateFormat.getTimeFormat(context)
        dateTextView.text = "${dateFormat.format(crime.date)}
                                                ${timeFormat.format(crime.date)}"
    }
}
...
```

Локалізація дати

Перенесення прив'язки даних до віджетів з CrimeAdapter до CrimeHolder

```
...

    private inner class CrimeAdapter(var crimes: List<Crime>) :
                                                RecyclerView.Adapter<CrimeHolder>() {
        override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): CrimeHolder{
            val view = layoutInflater.inflate(R.layout.list_item_crime, parent, false)
            return CrimeHolder(view)
        }
        override fun onBindViewHolder(holder: CrimeHolder, position: Int) {
            val crime = crimes[position]
            holder.bind(crime)
        }

        override fun getItemCount() = crimes.size
    }
}
```

Додання обробки події кліку на елементі списку RecyclerView

```
package com.example.criminalintent

import ...

private const val TAG = "CrimeListFragment"

class CrimeListFragment : Fragment() {
    private lateinit var crimeRecyclerView: RecyclerView
    private var adapter: CrimeAdapter? = null

    companion object {
        fun newInstance() = CrimeListFragment()
    }

    private val crimeListViewModel: CrimeListViewModel by lazy {
        ViewModelProvider(this).get(CrimeListViewModel::class.java)
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
    ): View? {...}

    private fun updateUI() { ... }
```

...

Додання обробки події кліку на елементі списку RecyclerView

```
...
private inner class CrimeHolder(view: View) : RecyclerView.ViewHolder(view),
                                                View.OnClickListener {
    private lateinit var crime: Crime
    private val titleTextView = itemView.findViewById(R.id.list_item_crime_title)
                                                as TextView
    private val dateTextView = itemView.findViewById(R.id.list_item_crime_date)
                                                as TextView

    init {
        itemView.setOnClickListener(this)
    }
    fun bind(crime: Crime) {
        this.crime = crime
        titleTextView.text = this.crime.title
        val dateFormat = DateFormat.getLongDateFormat(context)
        val timeFormat = DateFormat.getTimeFormat(context)
        dateTextView.text = "${dateFormat.format(crime.date)}
                                        ${timeFormat.format(crime.date)}"
    }
...
```

Додання обробки події кліку на елементі списку RecyclerView

```
...
        override fun onClick(v: View?) {
            Toast.makeText(context, "${crime.title} pressed!", Toast.LENGTH_SHORT)
                                                                    .show()
        }
    }
    private inner class CrimeAdapter(var crimes: List<Crime>) :
                                            RecyclerView.Adapter<CrimeHolder>() {

        ...
    }
}
```

Додання обробки події кліку на елементі списку RecyclerView

Рефакторинг методу обробки події кліку на елементі списку RecyclerView для виведення фрагменту деталізації

```
...
        override fun onClick(v: View?) {
//          Toast.makeText(context, "${crime.title} pressed!", Toast.LENGTH_SHORT)
//              .show()
            val fragment = CrimeFragment.newInstance()
            val fragmentManager = requireActivity().supportFragmentManager
            fragmentManager.beginTransaction()
                .replace(R.id.fragment_container, fragment)
                .addToBackStack(null)
                .commit()
        }
    }
    private inner class CrimeAdapter(var crimes: List<Crime>) :
                                        RecyclerView.Adapter<CrimeHolder>() {

        ...
    }
}
```