Програмування мобільних пристроїв

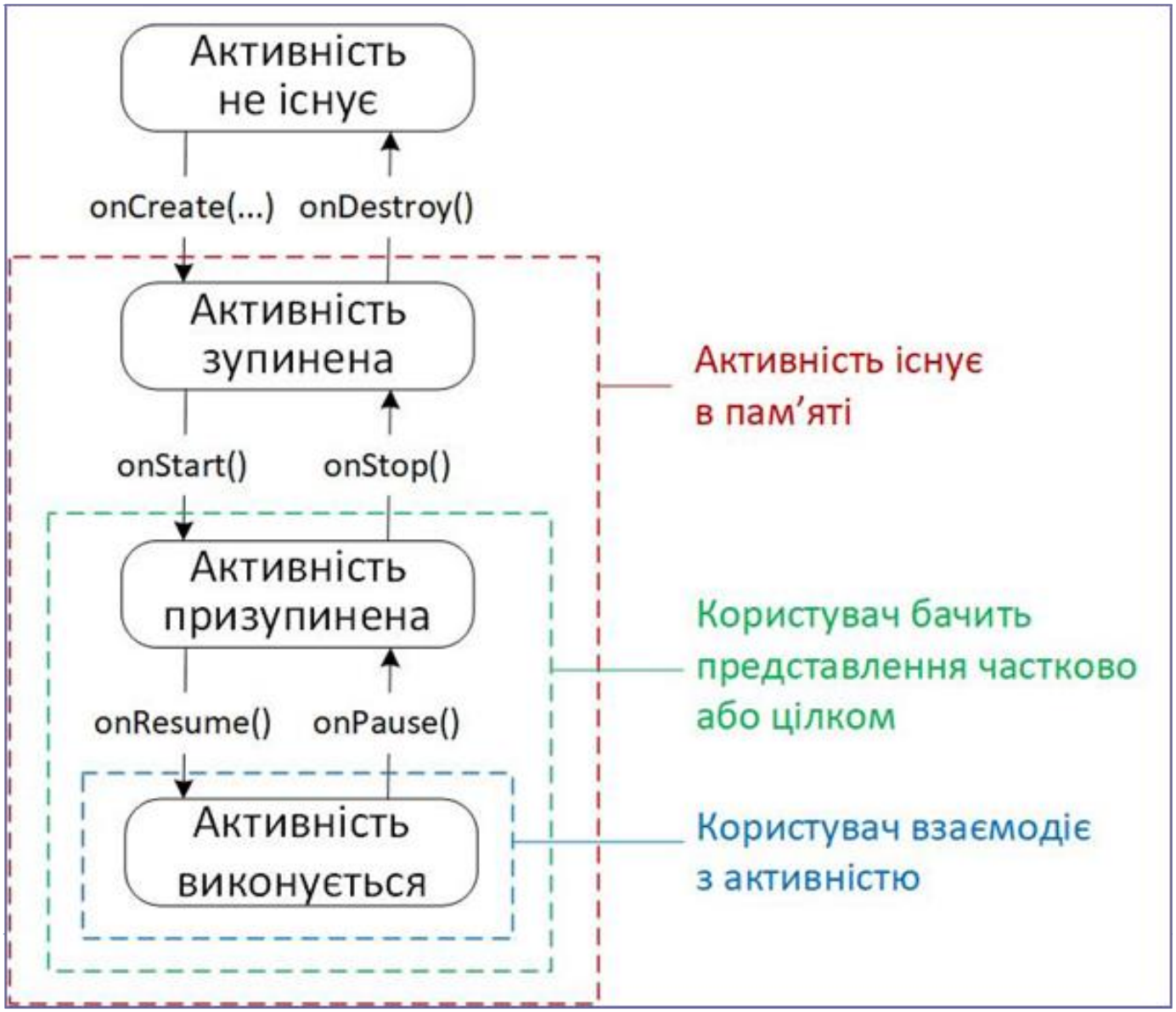
Життєвий цикл активності

Слайди до лекцій (4 змістовий модуль)

Життєвий цикл активності

- *Життєвий цикл активності* – це набір станів, у яких активність може перебувати протягом усього свого життя, від моменту її створення до моменту її знищення й відновлення виділених їй ресурсів системою.
- Під час життєвого циклу активність може знаходитись у чотирьох станах: виконання (Running), призупинення (Pause), зупинки (Stop) та неіснування (Destroy).
- Клас активності містить *функції зворотного виклику* (*callbacks*), які автоматично запускаються при прямих та зворотних переходах з кожного стану та які дозволяють запрограмувати необхідні дії, що будуть виконуватися при цих переходах.

Життєвий цикл активності - методи



Життєвий цикл активності - стани активності

Стан	Чи знаходиться у пам'яті?	Чи бачить користувач?	Чи знаходиться на передньому плані?
Активність не існує	Ні	Ні	Ні
Активність зупинена	Так	Ні	Ні
Активність призупинена	Так	Так/частково *	Ні
Активність виконується	Так	Так	Так

* Залежно від обставин призупинена активність може бути видима повністю або частково.

Життєвий цикл активності - стани активності

- *Неіснуючою* називається активність, яка ще не була запущена або щойно була знищена (наприклад, якщо користувач натиснув кнопку *Назад*). Іноді такі активності називають *знищеними*. При цьому активність відсутня у пам'яті і немає пов'язаного з нею представлення, яке користувач міг би бачити або з яким він міг би взаємодіяти.
- *Зупиненою* називається активність, яка знаходиться у пам'яті, але на екрані її не видно. Цей перехідний стан виникає, коли активність запускається, і повторюється щоразу, коли представлення активності повністю закрите (наприклад, коли користувач виводить іншу активність на передній план, натискає кнопку Головна або викликає список запущених програм для обрання необхідної.

Життєвий цикл активності - стани активності

- *Призупиненою* називається активність, представлення якої бачить користувач (можливо і не на передньому плані). Така ситуація виникає, наприклад, коли користувач запускає нове діалогове вікно або прозору активність. Також активність може бути повністю видимою, але бути не на передньому плані, якщо користувач переглядає дві активності у режимі розділеного екрана.
- *Виконуваною* називається активність, яка перебуває в пам'яті, цілком видима і знаходиться на передньому плані. Це та активність, з якою користувач взаємодіє у поточний момент. У будь-який момент часу лише одна активність у всій системі може перебувати у виконуваному стані. Це означає, що якщо одна активність переходить у виконуваний стан, то інша, швидше за все, призупиняється.

Життєвий цикл активності - функції зворотного виклику

Функції реєстрації подій android.util.Log

Рівень реєстрації	Функція	Пояснення
ERROR	Log.e(...)	Реєстрація помилок
WARNING	Log.w(...)	Реєстрація попереджень
INFO	Log.i(...)	Реєстрація інформаційних повідомлень
DEBUG	Log.d(...)	Реєстрація відлагоджувальних повідомлень
VERBOSE	Log.v(...)	Реєстрація повідомлень для розробника

```
public static int d(@Nullable String tag, @NonNull String msg)
```

```
public static int d(@Nullable String tag, @Nullable String msg,
                    @Nullable Throwable tr)
```

- Функція, що реєструє інформацію про виникнення виключення зазначеного типу

Життєвий цикл активності - функції зворотного виклику

```
private const val TAG = "MainActivity"
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        Log.d(TAG, "onCreate() called")
        ...
    }
    ...
    override fun onStart() {
        super.onStart()
        Log.d(TAG,"onStart() called")
    }
    override fun onResume() {
        super.onResume()
        Log.d(TAG,"onResume() called")
    }
...
```

Життєвий цикл активності - функції зворотного виклику

```
...
    override fun onPause() {
        super.onPause()
        Log.d(TAG,"onPause() called")
    }

    override fun onStop() {
        super.onStop()
        Log.d(TAG,"onStop() called")
    }

    override fun onDestroy() {
        super.onDestroy()
        Log.d(TAG,"onDestroy() called")
    }
}
```

Життєвий цикл активності - запуск застосунку

Logcat: Logcat

Nexus 5X (emulator-5554) Android 9, API 28

tag=:MainActivity

```
MainActivity    D  onCreate() called
MainActivity    D  onStart() called
MainActivity    D  onResume() called
```

Run | Logcat | App Quality Insights | Build | TODO | Problems | Terminal | Services

Життєвий цикл активності - запуск застосунку

Logcat: Logcat

Nexus 5X (emulator-5554) Android 9, API 28

tag=:MainActivity

```
MainActivity    D  onCreate() called
MainActivity    D  onStart() called
MainActivity    D  onResume() called
```

Run | Logcat | App Quality Insights | Build | TODO | Problems | Terminal | Services

Життєвий цикл активності - зупинка застосунку

Logcat: Logcat

Nexus 5X (emulator-5554) Android 9, API 28

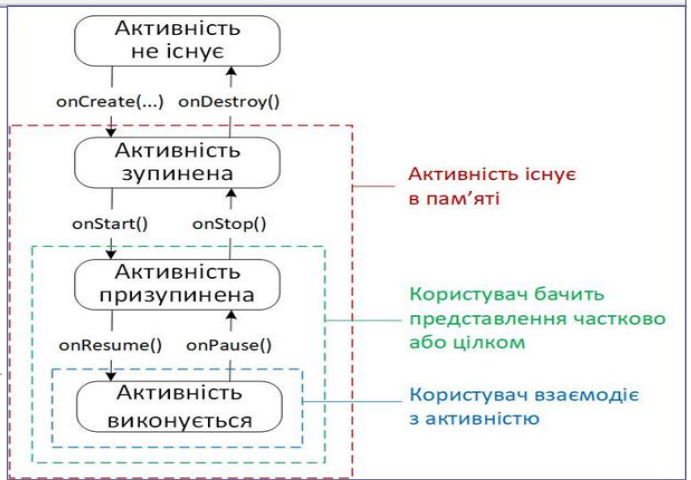
tag=:MainActivity

```
MainActivity  D  onCreate() called
MainActivity  D  onStart() called
MainActivity  D  onResume() called
MainActivity  D  onPause() called
MainActivity  D  onStop() called
```

Запуск

Головна (Home)

Run | Logcat | App Quality Insights | Build | TODO | Problems | Terminal | Services



Життєвий цикл активності - повернення до застосунку

Logcat: Logcat

Nexus 5X (emulator-5554) Android 9, API 28

tag=:MainActivity

```
MainActivity  D  onCreate() called
MainActivity  D  onStart() called
MainActivity  D  onResume() called
MainActivity  D  onPause() called
MainActivity  D  onStop() called
MainActivity  D  onStart() called
MainActivity  D  onResume() called
```

Запуск

Головна (Home)

Список - застосунок

Run | Logcat | App Quality Insights | Build | TODO | Problems | Terminal | Services

Життєвий цикл активності - видалення застосунку

Logcat: Logcat

Nexus 5X (emulator-5554) Android 9, API 28

tag=:MainActivity

```
MainActivity  D  onCreate() called
MainActivity  D  onStart() called
MainActivity  D  onResume() called
MainActivity  D  onPause() called
MainActivity  D  onStop() called
MainActivity  D  onStart() called
MainActivity  D  onResume() called
MainActivity  D  onPause() called
MainActivity  D  onStop() called
MainActivity  D  onDestroy() called
```

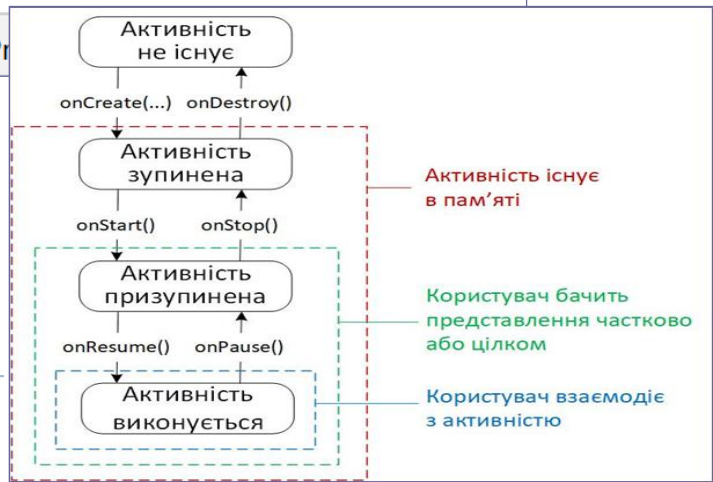
Запуск

Головна (Home)

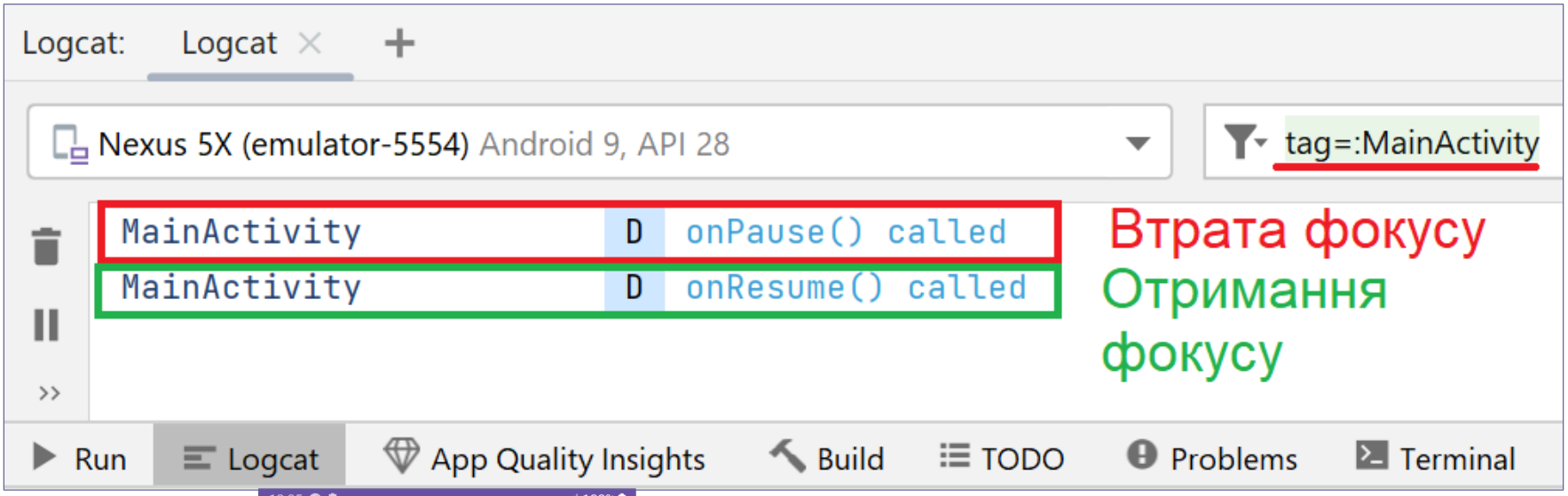
Список - застосунок

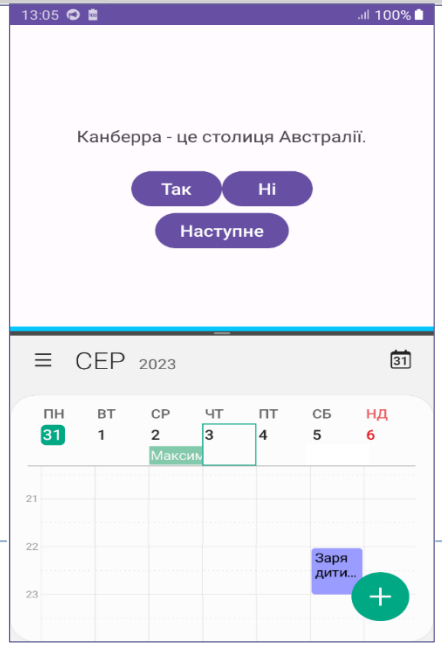
Назад

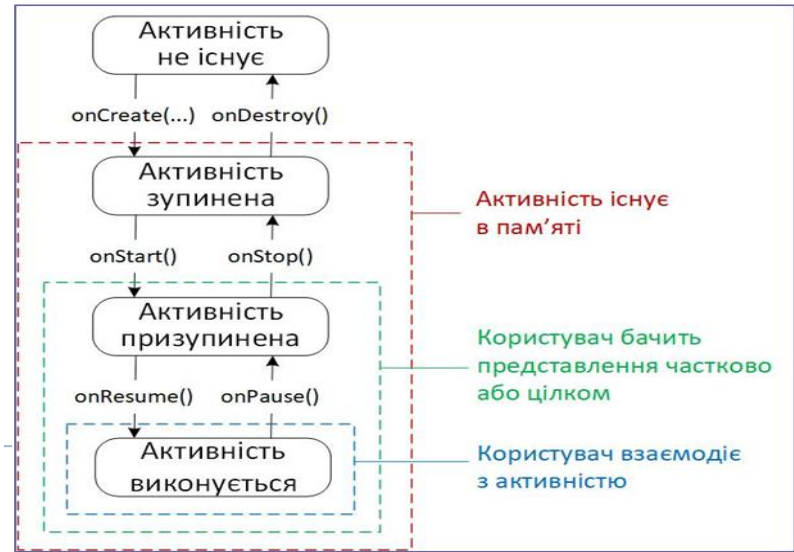
Run | Logcat | App Quality Insights | Build | TODO | Pr



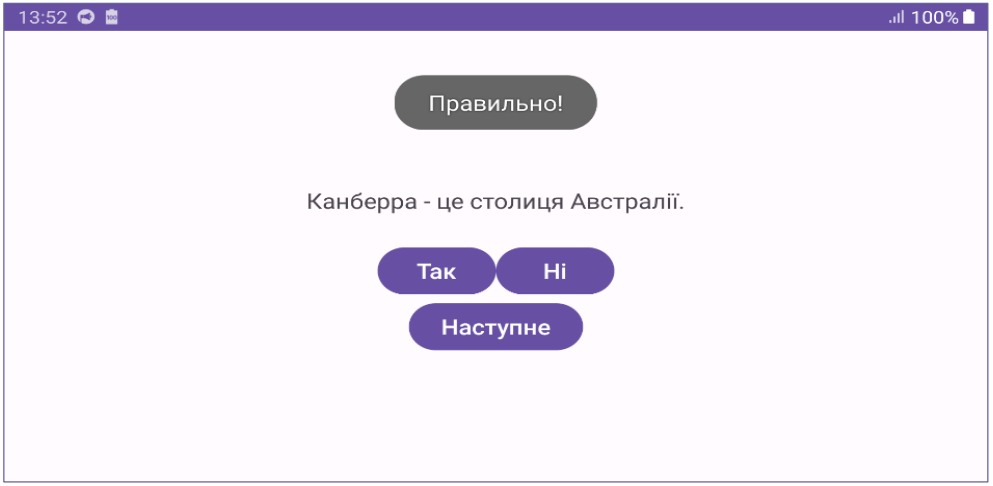
Життєвий цикл активності - втрата-отримання фокусу застосунком

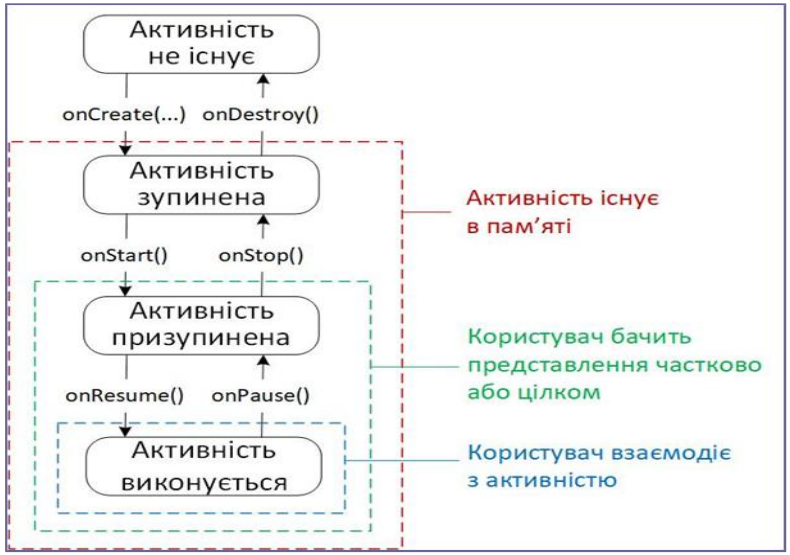


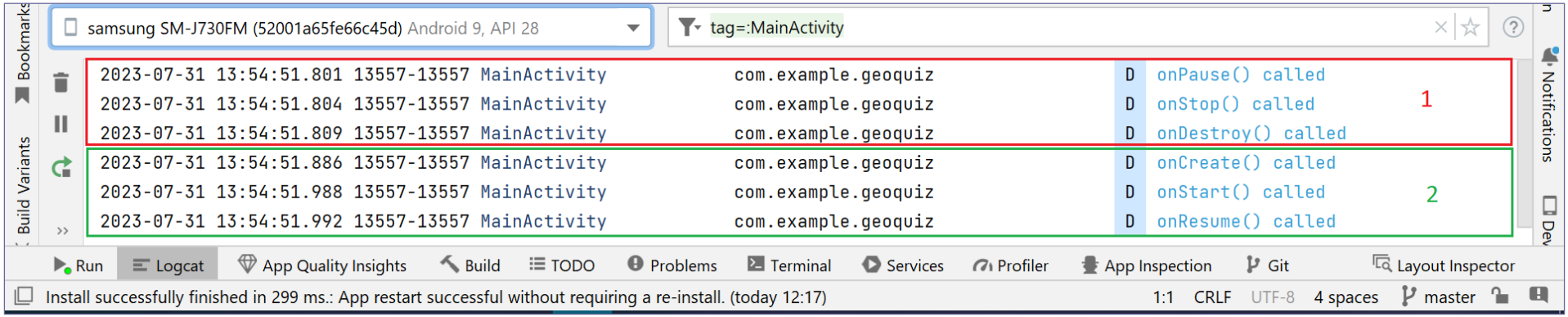




Поворот екрану пристрою - перезапуск активності





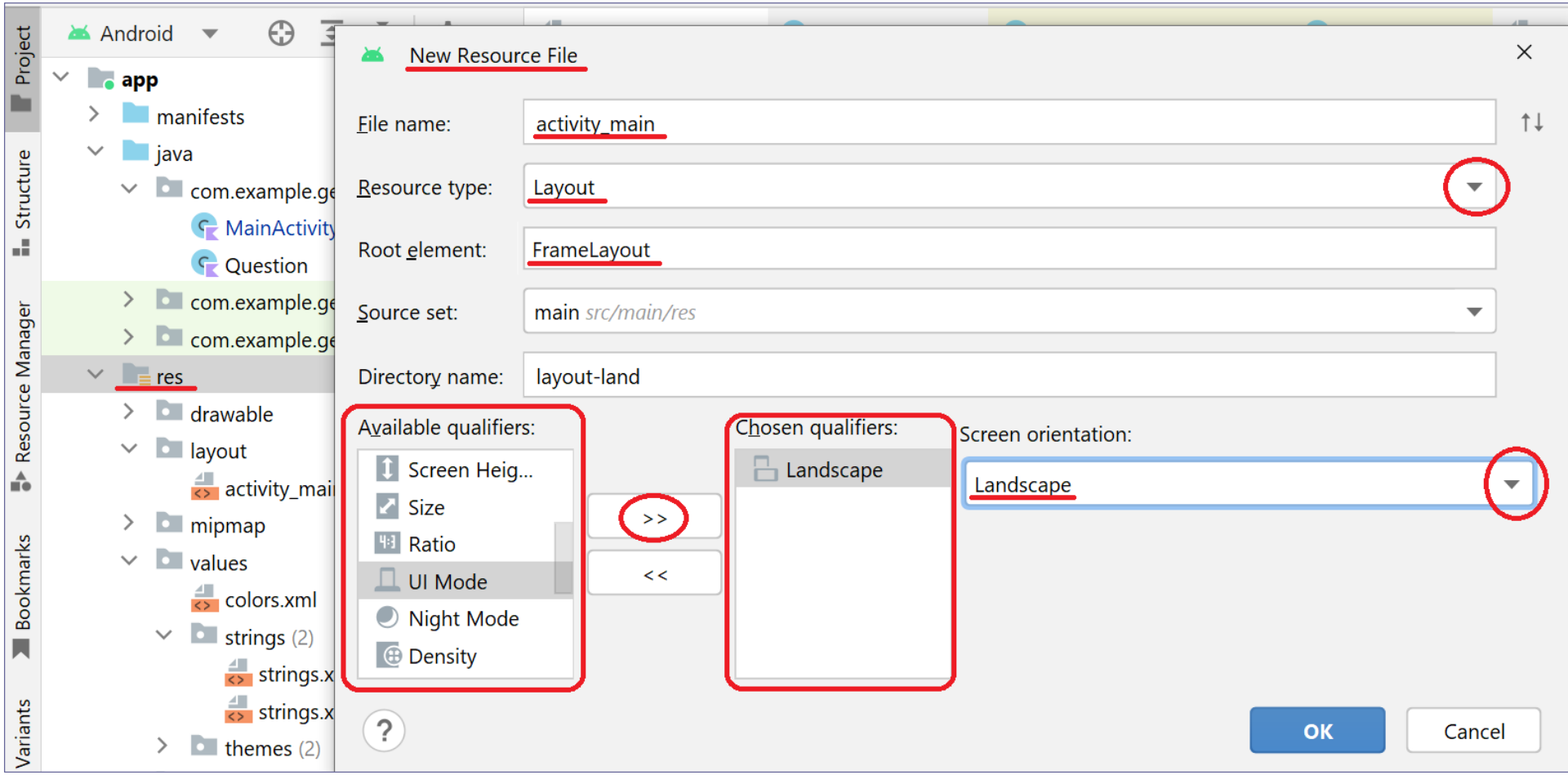


Конфігурація пристрою

- *Конфігурація пристрою* – це набір характеристик, що описують поточний стан пристрою, таких, як орієнтація екрану, щільність пікселів, розмір екрану, тип клавіатури, мова інтерфейсу тощо.
- Як правило, програми передбачають створення альтернативних ресурсів для різних конфігурацій пристроїв.
- При зміні конфігурації під час виконання може виявитися, що програма містить ресурси, які краще підходять для нової конфігурації. З цієї причини Android знищує активність, шукає ресурси, які найкраще підходять для нової конфігурації, і знову створює активність з цими ресурсами.

Створення макету для альбомної орієнтації

New–Android Resource File з контекстного меню каталогу *res*



Створення макету для альбомної орієнтації

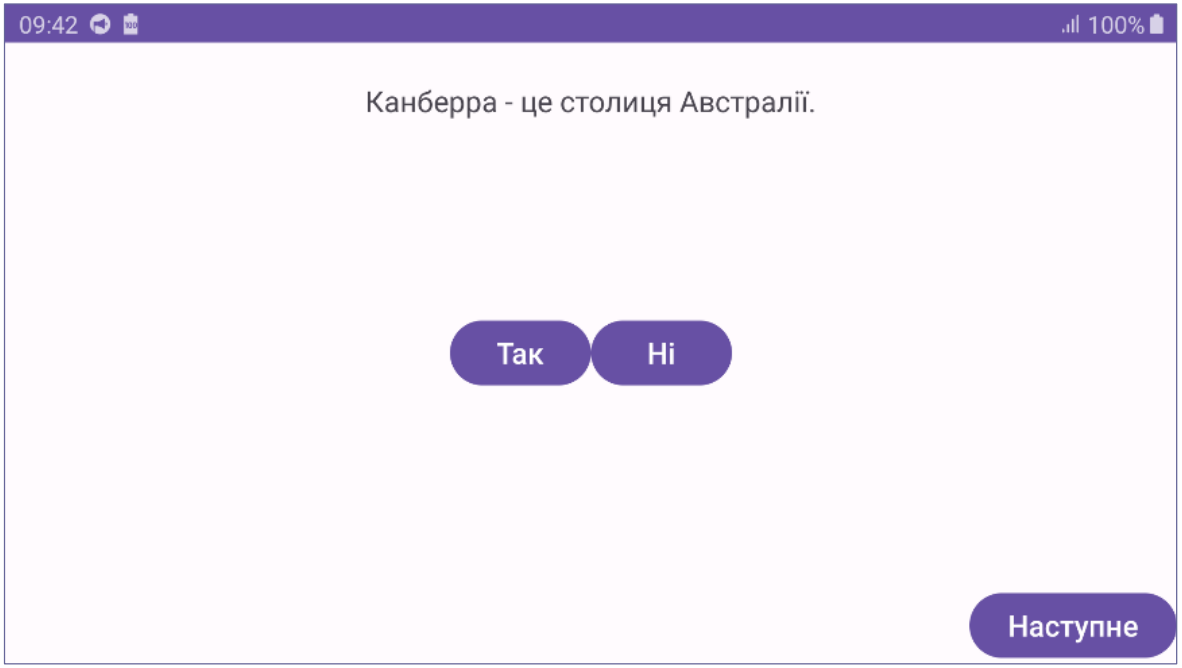
```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/question_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:padding="24dp"
        tools:text="@string/question_australia" />
    <!--        android:gravity="center"-->

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical|center_horizontal">
        <!--        android:orientation="horizontal"-->
...
```

дочірні віджети FrameLayout розміщуються відповідно до значення атрибутів android:layout_gravity

Створення макету для альбомної орієнтації

```
...
    <Button
        android:id="@+id/true_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/true_button" />

    <Button
        android:id="@+id/false_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/false_button" />
</LinearLayout>

<Button
    android:id="@+id/next_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/next_button"
    android:layout_gravity="bottom|right"/>

</FrameLayout>
```

Створення макету для альбомної орієнтації



Збереження стану активності

Наявні механізми збереження стану активності:

1) збереження у об'єкті androidx.lifecycle.ViewModel, який не видаляється при зміні конфігурації активності, але видаляється при завершенні процесу, у якому працює застосунок;
2) збереження у об'єкті android.os.Bundle у перевизначеному методі активності onSaveInstanceState(Bundle). Ця функція викликається щоразу, коли активність набуває стану *Активність зупинена*. Зчитуються збережені дані у функції активності OnCreate(Bundle?). Цей підхід зберігає усі дані активності (разом з додатковими визначеними програмістом), тому він є ресурсовимогливим до пристрою;
3) збереження у об'єкті androidx.lifecycle.ViewModel з обробником збереженого стану SavedStateHandle, який не видаляється при зміні конфігурації активності та перезапуску процесу, у якому працює застосунок.

Збереження стану активності у об'єкті класу ViewModel

- Об'єкт ViewModel пов'язаний з одним конкретним екраном і дозволяє вмістити всі дані про те, що має бути на екрані разом з бізнес-логікою зміни даних.
- Компоненти пакету androidx.lifecycle, зокрема ViewModel, спостерігають за життєвим циклом іншого компонента, наприклад Activity, та використовують цю інформацію у своїх цілях.

```
package com.example.geoquiz

import android.util.Log
import androidx.lifecycle.ViewModel

private const val TAG = "QuizViewModel"

class QuizViewModel : ViewModel() {
    init {
        Log.d(TAG, "ViewModel instance created")
    }
    override fun onCleared() {
        super.onCleared()
        Log.d(TAG, "ViewModel instance about be destroyed")
    }
}
```

Викликається перед знищенням об'єкту ViewModel

Збереження стану активності у об'єкті класу ViewModel

```
package com.example.geoquiz
import ...
private const val TAG = "MainActivity"
class MainActivity : AppCompatActivity() {
    ....
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        Log.d(TAG, "onCreate() called")
        setContentView(R.layout.activity_main)

        val provider:ViewModelProvider= ViewModelProvider(this)
        val quizViewModel=provider.get(QuizViewModel::class.java)
        Log.d(TAG,"Got a QuizViewModel: $quizViewModel")
...
        }
 }
```

Збереження стану активності у об'єкті класу ViewModel

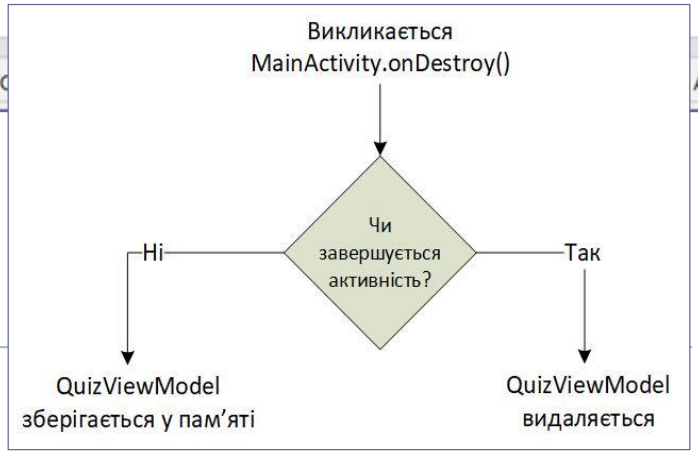
ogcat: Logcat

samsung SM-J730FM (52001a65fe66c45d) Android 9, API 28

tag=:MainActivity | tag=:QuizViewModel

```
2-9982  MainActivity     com.example.geoquiz   D  onCreate() called
2-9982  QuizViewModel    com.example.geoquiz   D  ViewModel instance created
2-9982  MainActivity     com.example.geoquiz   D  Got a QuizViewModel: com.example.geoquiz.QuizViewModel@7aee016
2-9982  MainActivity     com.example.geoquiz   D  onStart() called
2-9982  MainActivity     com.example.geoquiz   D  onResume() called                                              1
2-9982  MainActivity     com.example.geoquiz   D  onPause() called
2-9982  MainActivity     com.example.geoquiz   D  onStop() called                  Поворот екрану
2-9982  MainActivity     com.example.geoquiz   D  onDestroy() called
2-9982  MainActivity     com.example.geoquiz   D  onCreate() called
2-9982  MainActivity     com.example.geoquiz   D  Got a QuizViewModel: com.example.geoquiz.QuizViewModel@7aee016
2-9982  MainActivity     com.example.geoquiz   D  onStart() called
2-9982  MainActivity     com.example.geoquiz   D  onResume() called                                              2
2-9982  MainActivity     com.example.geoquiz   D  onPause() called
2-9982  MainActivity     com.example.geoquiz   D  onStop() called
2-9982  QuizViewModel    com.example.geoquiz   D  ViewModel instance about be destroyed
2-9982  MainActivity     com.example.geoquiz   D  onDestroy() called                                             3
```

Run | Logcat | App Quality Insights | Build | TO | App Inspection | Git | Layout Inspector



Збереження стану активності у об'єкті класу ViewModel - перенесення даних та логіки рівня моделі

```
package com.example.geoquiz

import android.util.Log
import androidx.lifecycle.ViewModel

private const val TAG = "QuizViewModel"

class QuizViewModel : ViewModel() {
    init {
        Log.d(TAG, "ViewModel instance created")
    }

    var currentIndex = 0

    private val questionBank = listOf(
        Question(R.string.question_australia, true),
        Question(R.string.question_oceans, true),
        Question(R.string.question_mideast, false),
        Question(R.string.question_africa, false),
        Question(R.string.question_americas, true),
        Question(R.string.question_asia, true)
    )
...
```

Збереження стану активності у об'єкті класу ViewModel - перенесення даних та логіки рівня моделі

```
...
    val currentQuestionAnswer: Boolean
        get() = questionBank[currentIndex].answer

    val currentQuestionText:Int
        get() = questionBank[currentIndex].textResId

    fun moveToNext(){
        currentIndex = (currentIndex + 1) % questionBank.size
    }

    override fun onCleared() {
        super.onCleared()
        Log.d(TAG, "ViewModel instance about be destroyed")
    }
}
```

Збереження стану активності у об'єкті класу ViewModel - рефакторинг MainActivity

```
package com.example.geoquiz
import ...
private const val TAG = "MainActivity"
class MainActivity : AppCompatActivity() {
    private lateinit var trueButton: Button
    ...
    private val quizViewModel: QuizViewModel by lazy {
        ViewModelProvider(this,
                          defaultViewModelProviderFactory)[QuizViewModel::class.java]
    }
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        ...
        nextButton.setOnClickListener {
            quizViewModel.moveToNext()
            updateQuestion()
        }
        updateQuestion()     }  ...
```

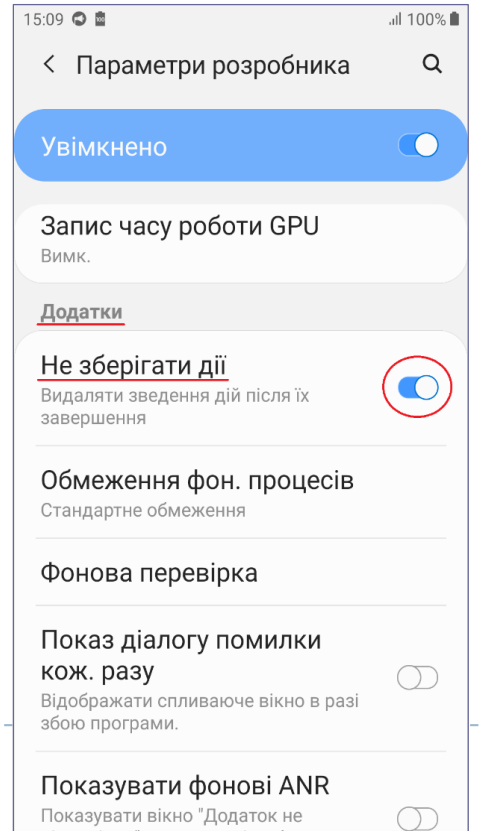
Збереження стану активності у об'єкті класу ViewModel - рефакторинг MainActivity

```
...
    private fun updateQuestion() {
        val questionTextResId = quizViewModel.currentQuestionText
        questionTextView.setText(questionTextResId)
    }
    private fun checkAnswer(userAnswer: Boolean): Int {
        val correctAnswer = quizViewModel.currentQuestionAnswer
        return if (userAnswer == correctAnswer) R.string.correct_toast
                        else R.string.incorrect_toast
    }
...
}
```

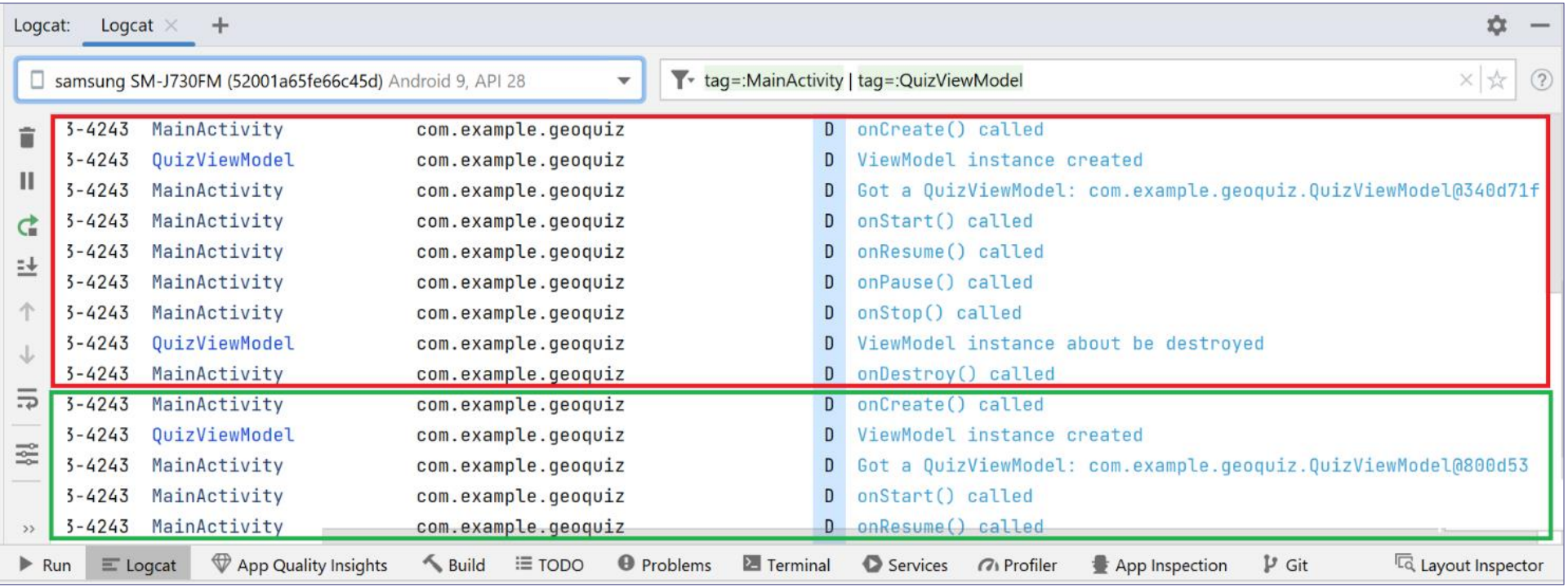
Застосування ModelView наблизило архітектуру застосунку до шаблону MVC: тепер дані та їх логіка винесені на рівень моделі, а в активності залишилися функції, притаманні рівню контролера

Збереження даних після завершення процесу

- Процеси, що містять активності у станах *Активність виконується* або *Активність призупинена*, мають високий пріоритет, а процеси з активностями у стані *Активність зупинена* – більш низький (Рис. 34). Коли операційній системі необхідно звільнити ресурси, вона спочатку завершуватиме процеси з низьким пріоритетом.



Збереження даних після завершення процесу



- Необхідно забезпечити збереження стану застосунку перед перезапуском його процесу так, щоб користувач взагалі не помічав, що активність та ресурси застосунку були видалені та створені наново.

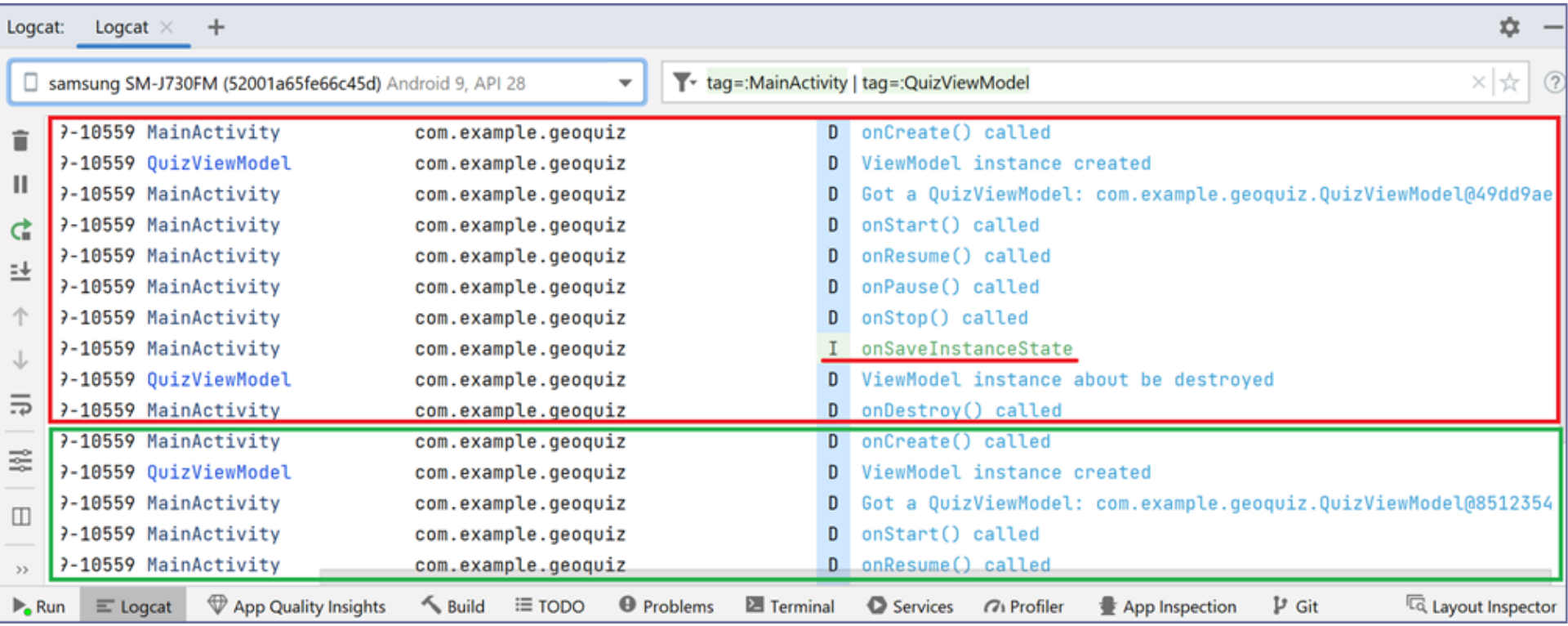
Збереження даних після завершення процесу у вигляді збереженого стану екземпляра

- Дані можна зберігати як *збережений стан екземпляра* – savedInstanceState: Bundle шляхом перевизначення методу Activity onSaveInstanceState(Bundle).
- Операційна система викликає цей метод щоразу, коли активність набуває стану *Активність зупинена*.
- Реалізація за замовчуванням onSaveInstanceState(Bundle) зберігає всі представлення та стан активності в об'єкті savedInstanceState: Bundle.
- savedInstanceState: Bundle - це структура даних, яка зберігає пари "ключ–значення", де ключі є рядками, а значення – об'єктами.
- Можна перевизначити onSaveInstanceState(Bundle) для збереження у об'єкті savedInstanceState: Bundle додаткових даних, які потім можуть бути зчитані назад функцією onCreate(savedInstanceState: Bundle?) при запуску активності.

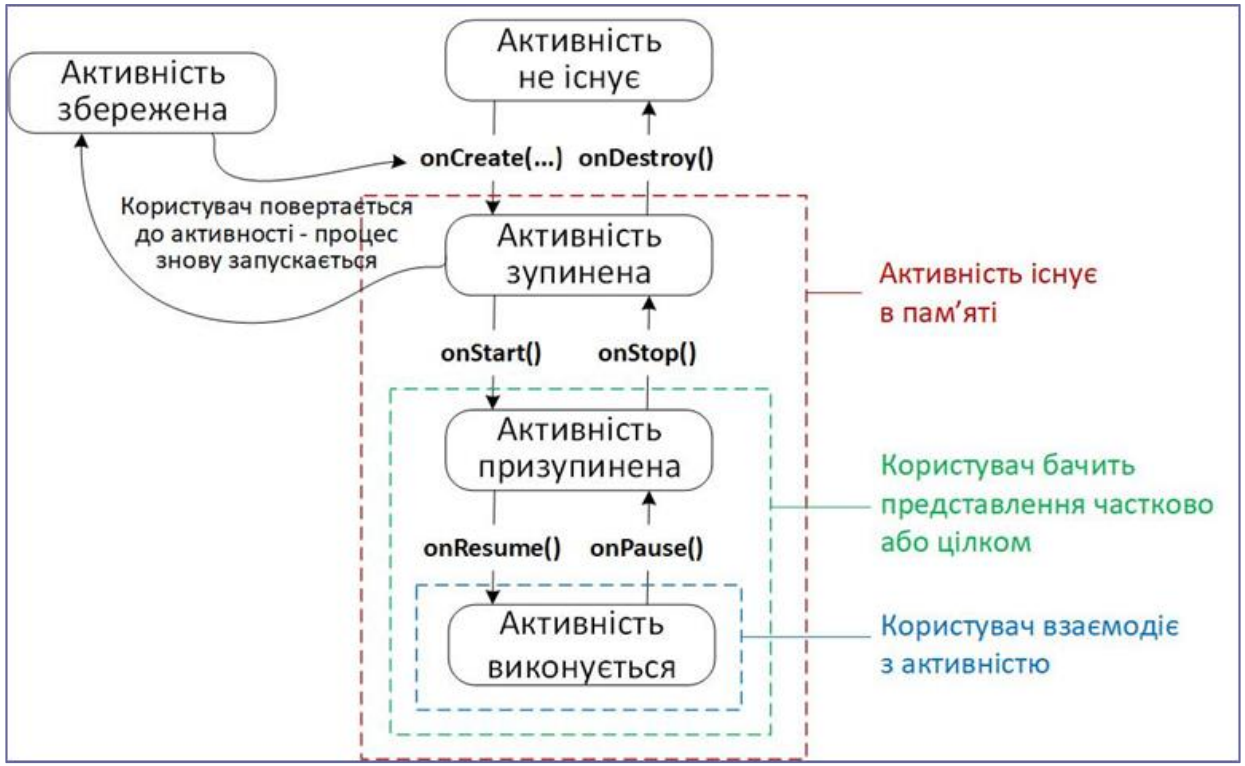
Збереження даних після завершення процесу у вигляді збереженого стану екземпляра

```
package com.example.geoquiz

import ...

private const val TAG = "MainActivity"
private const val KEY_INDEX = "index"

class MainActivity : AppCompatActivity() {
...
    override fun onSaveInstanceState(savedInstanceState: Bundle) {
        super.onSaveInstanceState(savedInstanceState)
        Log.i(TAG, "onSaveInstanceState")
        savedInstanceState.putInt(KEY_INDEX,
                quizViewModel.currentIndex)
    }
...
    override fun onCreate(savedInstanceState: Bundle?) {
        ...
        setContentView(R.layout.activity_main)
        Log.d(TAG, "Got a QuizViewModel: $quizViewModel")
        val currentIndex =
            savedInstanceState?.getInt(KEY_INDEX, 0) ?: 0
        quizViewModel.currentIndex=currentIndex   ...    } ... }
```

Збереження даних після завершення процесу у вигляді збереженого стану екземпляра



Збереження даних після завершення процесу у вигляді збереженого стану екземпляра



- Перевизначена функція onSaveInstanceState(Bundle) зберігає, як правило, невеликі перехідні дані, які стосуються поточної активності.
- Перевизначена функція OnStop() зберегає, як правило, постійні дані, такі як налаштування користувача, активність може бути завершена в будь-який момент після завершення цієї функції.

Збереження даних у об'єкті ViewModel із модулем Saved State

- Враховуючи, що використання ViewModel забезпечує розділення функцій відповідно до архітектури MVC та забезпечує ефективне використання ресурсів мобільного пристрою, починаючи з версії Activity 1.1.0 був доданий конструктор ViewModel, який приймає об'єкт SavedStateHandle, за допомогою якого можна зберігати та отримувати необхідні дані у модулі Saved State об'єкта ViewModel, який зберігається навіть після видалення ViewModel разом з активністю.

Збереження даних у об'єкті ViewModel із модулем Saved State - рефакторинг QuizViewModel

```
package com.example.geoquiz

import ...

private const val TAG = "QuizViewModel"
private const val KEY_INDEX = "index"

class QuizViewModel(state: SavedStateHandle) : ViewModel() {
    ...
    private val savedStateHandle = state
    private var currentIndex = getCurrentIndex()
    ...
    fun moveToNext() {
        currentIndex = (currentIndex + 1) % questionBank.size
        saveCurrentIndex(currentIndex)
    }
    private fun saveCurrentIndex(currentIndex: Int) {
        // Sets a new value for the object associated to the key.
        Log.i(TAG, "Current index saved in $this")
        savedStateHandle[KEY_INDEX] = currentIndex
    }
...
```

Перенесена з MainActivity

Конструктор приймає SavedStateHandle

Додана

Отримуємо поточний індекс

Зберігаємо поточний індекс у SavedStateHandle

Збереження даних у об'єкті ViewModel із модулем Saved State - рефакторинг QuizViewModel

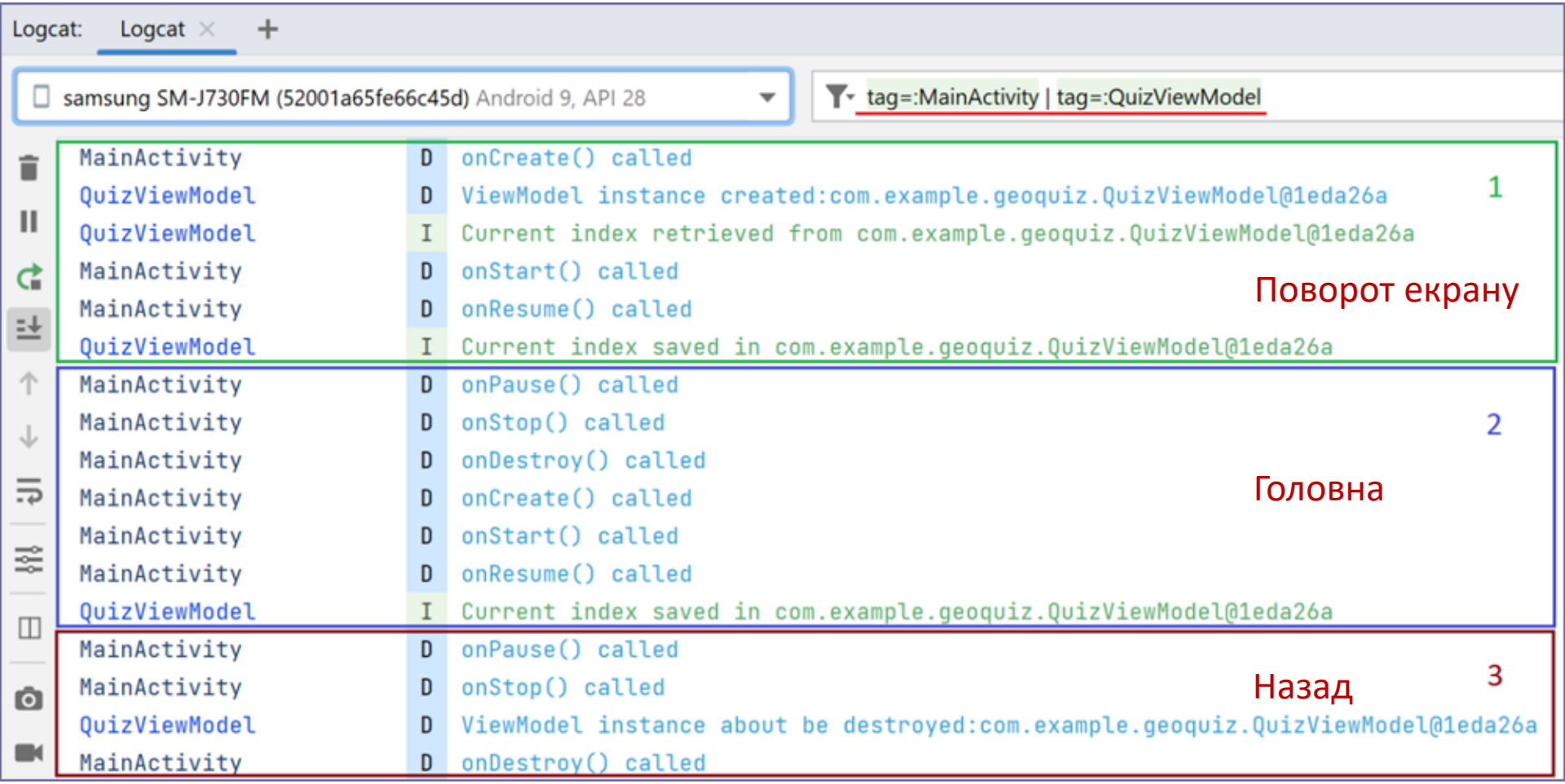
```
...
    private fun getCurrentIndex(): Int {    Отримуємо поточний індекс з SavedStateHandle
        // Gets the current value of the user id from the saved state handle
        Log.i(TAG, "Current index retrieved from $this")
        return savedStateHandle[KEY_INDEX] ?: 0
    }
    ...
}
```

- з класу MainActivity можна видалити константу-ключ KEY_INDEX = "index", перевизначену функцію onSaveInstanceState(savedInstanceState: Bundle) та видалити з функції onCreate(savedInstanceState: Bundle?) код отримання та оновлення поточного індексу питань – тепер ця функція перейшла до класу QuizViewModel, таки чином, логіка, пов'язана з даними тепер повністю винесена з активності до модельного класу.
- Отримання посилання на об'єкт quizViewModel тепер використовує конструктор ViewModelProvider, який окрім посилання на поточну активність, приймає фабрику об'єктів ViewModelProvider, які тепер підтримують роботу з модулем Saved State.

Збереження даних у об'єкті ViewModel із модулем Saved State - рефакторинг MainActivity

```
package com.example.geoquiz

import ...

private const val TAG = "MainActivity"

class MainActivity : AppCompatActivity() {
    ...
    private lateinit var questionTextView: TextView
    private val quizViewModel: QuizViewModel by lazy {
        ViewModelProvider(this,
                          defaultViewModelProviderFactory)[QuizViewModel::class.java]
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        ...
    }
//    override fun onSaveInstanceState(savedInstanceState: Bundle) {
//        super.onSaveInstanceState(savedInstanceState)
//        Log.i(TAG, "onSaveInstanceState")
//        savedInstanceState.putInt(KEY_INDEX, quizViewModel.currentIndex)
//    } ... }
```

Збереження даних у об'єкті ViewModel із модулем Saved State



Обов'язково відключіть опцію *Не зберігати дії* у *Параметрах розробника*, у противному разі буде знижена продуктивність роботи пристрою.

Android Jetpack Components

- Клас androidx.lifecycle.ViewModel входить до бібліотеки з набору компонентів *Android Jetpack Components* або *Jetpack*. Це набір бібліотек, створених Google, щоб спростити різноманітні аспекти розробки на Android (Jetpack). Кожна Jetpack-бібліотека знаходиться в пакеті, який починається з androidx. Тому терміни *AndroidX* і *Jetpack* використовуються як синоніми.
- Jetpack-бібліотеки розбиті на чотири категорії:
  1) *фундамент*,
  2) *архітектура*,
  3) *поведінка*,
  4) *інтерфейс користувача*.

Android Jetpack Components

- ViewModel один з компонентів *архітектури*.
- Прикладами основних компонентів *фундаментальних бібліотек* можна назвати AppCompat (зверніть увагу, що MainActivity успадковується від androidx.appcompat.app.AppCompatActivity), Test (використовується при тестуванні) та Android KTX (набір розширень мови програмування Kotlin).
- Прикладом компонентів *поведінкових бібліотек* можна назвати Notifications,
- Прикладом компонентів *бібліотек інтерфейса користувача* – ConstraintLayout.
- На сьогодні при створенні проєктів у AndroidStudio компоненти бібліотек AndroidX підключаються автоматично (дивись файл build.gradle модуля app).