LAPORAN PRAKTIKUM
PEMROGRAMAN MOBILE
MODUL 4

VIEWMODEL AND DEBUGGING

Oleh:

Putra Whyra Pratama S. **NIM. 2310817210029**

PROGRAM STUDI TEKNOLOGI INFORMASI
FAKULTAS TEKNIK
UNIVERSITAS LAMBUNG MANGKURAT
MEI 2025

LEMBAR PENGESAHAN
LAPORAN PRAKTIKUM PEMROGRAMAN I
MODUL 4

Laporan Praktikum Pemrograman Mobile Modul 4: ViewModel and Debugging ini disusun sebagai syarat lulus mata kuliah Praktikum Pemrograman Mobile. Laporan Prakitkum ini dikerjakan oleh:

Nama Praktikan : Putra Whyra Pratama S.
NIM : 2310817210029

Menyetujui, Asisten Praktikum	Mengetahui, Dosen Penanggung Jawab Praktikum
Zulfa Auliya Akbar NIM. 2210817210026	Muti`a Maulida S.Kom M.T.I NIP. 19881027 201903 20 13

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

SOAL 1

Soal Praktikum:

Lanjutkan aplikasi Android berbasis XML dan Jetpack Compose yang sudah dibuat pada Modul 3 dengan menambahkan modifikasi sesuai ketentuan berikut:

a. Buatlah sebuah ViewModel untuk menyimpan dan mengelola data dari list item. Data tidak boleh disimpan langsung di dalam Fragment atau Activity.
b. Gunakan ViewModelFactory dalam pembuatan ViewModel
c. Gunakan StateFlow untuk mengelola event onClick dan data list item dari ViewModel ke Fragment
d. gunakan logging untuk event berikut:
    a. Log saat data item masuk ke dalam list
    b. Log saat tombol Detail dan tombol Explicit Intent ditekan
    c. Log data dari list yang dipilih ketika berpindah ke halaman Detail
e. Gunakan tool Debugger di Android Studio untuk melakukan debugging pada aplikasi. Cari setidaknya satu breakpoint yang relevan dengan aplikasi. Lalu, gunakan fitur Step Into, Step Over, dan Step Out. Setelah itu, jelaskan fungsi Debugger, cara menggunakan Debugger, serta fitur Step Into, Step Over, dan Step Out

A. Source Code

1. MainActivity.kt

Tabel 1. Source Code MainActivity.kt

```
1   package com.example.wuwalist
2
3   import android.os.Bundle
4   import android.util.Log
5   import androidx.activity.enableEdgeToEdge
6   import androidx.appcompat.app.AppCompatActivity
7   import androidx.core.view.ViewCompat
8   import androidx.core.view.WindowInsetsCompat
9
10  class MainActivity : AppCompatActivity() {
11      override fun onCreate(savedInstanceState: Bundle?) {
12          super.onCreate(savedInstanceState)
13          setContentView(R.layout.activity_main)
14
15          val fragmentManager = supportFragmentManager
16          val homeFragment = HomeFragment()
17          val fragment =
    fragmentManager.findFragmentByTag(HomeFragment::class.java.s
    impleName)
18          if (fragment !is HomeFragment) {
19              Log.d("MyFlexibleFragment", "Fragment Name :" +
    HomeFragment::class.java.simpleName)
20              fragmentManager
21                  .beginTransaction()
22                  .add(R.id.frame_container, homeFragment,
    HomeFragment::class.java.simpleName)
23                  .commit()
24          }
25      }
26  }
```

2. Character.kt

Tabel 2. Source Code Character.kt

```
1   package com.example.wuwalist
2
3   import android.os.Parcelable
4   import kotlinx.parcelize.Parcelize
5
6
7   @Parcelize
8   data class Character(
9       val name: String,
10      val link: String,
11      val description: String,
12      val photo: Int
```

```
13  ):Parcelable
```

3. ListCharacterAdapter.kt

Tabel 3. Source Code ListCharacterAdapter.kt

```
1   package com.example.wuwalist.adapter
2
3   import android.view.LayoutInflater
4   import android.view.View
5   import android.view.ViewGroup
6   import android.widget.Button
7   import android.widget.ImageView
8   import android.widget.TextView
9   import androidx.recyclerview.widget.RecyclerView
10  import com.example.wuwalist.model.Character
11  import com.example.wuwalist.R
12
13  class ListCharacterAdapter(
14      private val listCharacter: ArrayList<Character>,
15      private val onWikiClick: (String) -> Unit,
16      private val onDetailClick: (String, Int, String) ->
    Unit)
17      :
    RecyclerView.Adapter<ListCharacterAdapter.ListViewHolder>()
    {
18
19      class ListViewHolder(itemView: View) :
    RecyclerView.ViewHolder(itemView) {
20          val imgPhoto: ImageView =
    itemView.findViewById(R.id.img_character)
21          val tvName: TextView =
    itemView.findViewById(R.id.tv_character)
22          val tvDeskripsi: TextView =
    itemView.findViewById(R.id.tv_deskripsi)
23          val btnWiki: Button =
    itemView.findViewById(R.id.btn_link)
24          val btnDetail: Button =
    itemView.findViewById(R.id.btn_detail)
25      }
26
27      override fun onCreateViewHolder(parent: ViewGroup,
    viewType: Int): ListViewHolder {
28          val view: View =
    LayoutInflater.from(parent.context).inflate(R.layout.item_ch
    aracter, parent, false)
29          return ListViewHolder(view)
30      }
31
32      override fun getItemCount(): Int = listCharacter.size
33
```

```
34      override fun onBindViewHolder(holder: ListViewHolder,
    position: Int) {
35          val (name, link, description, photo) =
    listCharacter[position]
36          holder.tvName.text = name
37          holder.imgPhoto.setImageResource(photo)
38          holder.tvDeskripsi.text = description
39          holder.btnWiki.setOnClickListener {
    onWikiClick(link) }
40          holder.btnDetail.setOnClickListener {
    onDetailClick(name, photo, description) }
41      }
42      fun setData(newList: List<Character>) {
43          listCharacter.clear()
44          listCharacter.addAll(newList)
45      }
46  }
```

4. HomeFragment.kt

Tabel 4. Source Code HomeFragment.kt

```
1   package com.example.wuwalist.presentation.home
2
3   import android.content.Intent
4   import android.net.Uri
5   import android.os.Bundle
6   import android.util.Log
7   import androidx.fragment.app.Fragment
8   import androidx.recyclerview.widget.LinearLayoutManager
9   import android.view.LayoutInflater
10  import android.view.View
11  import android.view.ViewGroup
12  import androidx.fragment.app.viewModels
13  import androidx.lifecycle.lifecycleScope
14  import com.example.wuwalist.R
15  import com.example.wuwalist.adapter.ListCharacterAdapter
16  import com.example.wuwalist.databinding.FragmentHomeBinding
17  import com.example.wuwalist.utils.HomeViewModelFactory
18  import
    com.example.wuwalist.presentation.detail.DetailFragment
19  import kotlinx.coroutines.launch
20  import kotlinx.coroutines.flow.collectLatest
21  import java.util.ArrayList
22
23  class HomeFragment : Fragment() {
24
25      private var _binding: FragmentHomeBinding? = null
26      private val binding get() = _binding!!
27
```

```
28      private lateinit var characterAdapter:
    ListCharacterAdapter
29
30      private val viewModel: HomeViewModel by viewModels {
31          HomeViewModelFactory(resources)
32      }
33
34      override fun onCreateView(
35          inflater: LayoutInflater, container: ViewGroup?,
36          savedInstanceState: Bundle?
37      ): View {
38          _binding = FragmentHomeBinding.inflate(inflater,
    container, false)
39          return binding.root
40      }
41
42      override fun onViewCreated(view: View,
    savedInstanceState: Bundle?) {
43          super.onViewCreated(view, savedInstanceState)
44
45          setupRecyclerView()
46          observeCharacterList()
47
48          viewModel.loadCharacters()
49      }
50
51      private fun setupRecyclerView() {
52          characterAdapter = ListCharacterAdapter(
53              ArrayList(),
54              onWikiClick = { link ->
55
56                  Log.e("Explicit intent to Web","Going to
    $link")
57
58                  val intent = Intent(Intent.ACTION_VIEW,
    Uri.parse(link))
59                  startActivity(intent)
60
61              },
62              onDetailClick = { name, photo, description ->
63                  val detailFragment = DetailFragment().apply
    {
64
65
66
67                      Log.e("Move to detail page","move to
    $name detail page")
68
69                      arguments = Bundle().apply {
70                          putString("EXTRA_NAME", name)
71                          putInt("EXTRA_PHOTO", photo)
```

```
72                          putString("EXTRA_DESCRIPTION",
    description)
73                      }
74                  }
75
76                  parentFragmentManager.beginTransaction()
77                      .replace(R.id.frame_container,
    detailFragment)
78                      .addToBackStack(null)
79                      .commit()
80              }
81          )
82
83          binding.rvCharacter.apply {
84              layoutManager = LinearLayoutManager(context)
85              adapter = characterAdapter
86              setHasFixedSize(true)
87          }
88      }
89
90      private fun observeCharacterList() {
91          viewLifecycleOwner.lifecycleScope.launch {
92              viewModel.characterList.collectLatest { list ->
93                  characterAdapter.setData(list)
94              }
95          }
96      }
97
98      override fun onDestroyView() {
99          super.onDestroyView()
100         _binding = null
101     }
102 }
```

5. DetailFragment.kt

Tabel 5. Source Code DetailFragment.kt

```
1   package com.example.wuwalist
2
3   import android.os.Bundle
4   import androidx.fragment.app.Fragment
5   import android.view.LayoutInflater
6   import android.view.View
7   import android.view.ViewGroup
8   import
    com.example.wuwalist.databinding.FragmentDetailBinding
9
10  class DetailFragment : Fragment() {
11
12      private var _binding: FragmentDetailBinding? = null
13      private val binding get() = _binding!!
14
15      override fun onCreateView(
16          inflater: LayoutInflater, container: ViewGroup?,
17          savedInstanceState: Bundle?
18      ): View {
19          _binding = FragmentDetailBinding.inflate(inflater,
    container, false)
20
21          val name = arguments?.getString("EXTRA_NAME")
22          val photo = arguments?.getInt("EXTRA_PHOTO")
23          val description =
    arguments?.getString("EXTRA_DESCRIPTION")
24
25
26          binding.tvCharacter.text = name
27          binding.tvDeskripsi.text = description
28          photo?.let {
29              binding.imgCharacter.setImageResource(it)
30          }
31
32          return binding.root
33      }
34
35      override fun onDestroyView() {
36          super.onDestroyView()
37          _binding = null
38      }
39  }
```

6. activity_main.xml

Tabel 6. Source Code activity_main.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <FrameLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      tools:context=".MainActivity"
8      android:id="@+id/frame_container">
9  </FrameLayout>
```

7. fragment_home.xml

Tabel 7. Source Code fragment_home.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      android:layout_width="wrap_content"
5      android:layout_height="wrap_content"
6      xmlns:tools="http://schemas.android.com/tools"
7      android:orientation="horizontal"
8      tools:context=".HomeFragment">
9
10     <androidx.recyclerview.widget.RecyclerView
11         android:id="@+id/rv_character"
12         android:layout_width="match_parent"
13         android:layout_height="match_parent"
14         app:layout_constraintBottom_toBottomOf="parent"
15         app:layout_constraintEnd_toEndOf="parent"
16         app:layout_constraintStart_toStartOf="parent"
17         app:layout_constraintTop_toTopOf="parent" />
18 </androidx.constraintlayout.widget.ConstraintLayout>
```

8. fragment_detail.xml

Tabel 8. Source Code fragment_detail.xml

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
3       xmlns:app="http://schemas.android.com/apk/res-auto"
4       xmlns:tools="http://schemas.android.com/tools"
5       android:layout_width="match_parent"
6       android:layout_height="match_parent"
7       tools:context=".DetailFragment">
8
9       <ImageView
10          android:id="@+id/img_character"
11          android:layout_width="190dp"
12          android:layout_height="261dp"
13          android:layout_marginTop="16dp"
14          android:src="@drawable/card_carlotta"
15          app:layout_constraintEnd_toEndOf="parent"
16          app:layout_constraintStart_toStartOf="parent"
17          app:layout_constraintTop_toTopOf="parent" />
18
19      <TextView
20          android:id="@+id/tv_character"
21          android:layout_width="wrap_content"
22          android:layout_height="wrap_content"
23          android:layout_marginTop="19dp"
24          android:text="TextView"
25          app:layout_constraintEnd_toEndOf="parent"
26          app:layout_constraintStart_toStartOf="parent"
27
    app:layout_constraintTop_toBottomOf="@+id/img_character" />
28
29      <TextView
30          android:id="@+id/tv_profile"
31          android:layout_width="wrap_content"
32          android:layout_height="wrap_content"
33          android:layout_marginStart="10dp"
34          android:gravity="start"
35          android:text="Profile"
36          android:textStyle="bold"
37          app:layout_constraintStart_toStartOf="parent"
38
    app:layout_constraintTop_toBottomOf="@+id/tv_character" />
39
40      <TextView
41          android:id="@+id/tv_deskripsi"
42          android:layout_width="wrap_content"
43          android:layout_height="wrap_content"
```

```
44          android:layout_marginStart="10dp"
45          android:layout_marginTop="20dp"
46          android:text="TextView"
47          app:layout_constraintStart_toStartOf="parent"
48
    app:layout_constraintTop_toBottomOf="@+id/tv_profile" />
49  </androidx.constraintlayout.widget.ConstraintLayout>
```

9. item_character.xml

Tabel 9. Source Code item_character.xml

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
3       xmlns:app="http://schemas.android.com/apk/res-auto"
4       xmlns:tools="http://schemas.android.com/tools"
5       android:layout_width="match_parent"
6       android:layout_height="wrap_content">
7
8       <androidx.cardview.widget.CardView
9           android:id="@+id/card_view"
10          android:layout_width="wrap_content"
11          android:layout_height="wrap_content"
12          android:layout_marginStart="20dp"
13          android:layout_marginTop="10dp"
14          android:layout_marginEnd="20dp"
15          android:layout_marginBottom="10dp"
16          app:cardBackgroundColor="#FFFFFF"
17          app:cardCornerRadius="16dp"
18          app:cardElevation="16dp"
19          app:cardPreventCornerOverlap="false"
20          app:layout_constraintBottom_toBottomOf="parent"
21          app:layout_constraintEnd_toEndOf="parent"
22          app:layout_constraintStart_toStartOf="parent"
23          app:layout_constraintTop_toTopOf="parent">
24
25          <androidx.constraintlayout.widget.ConstraintLayout
26              android:layout_width="match_parent"
27              android:layout_height="match_parent"
28              android:layout_marginStart="8dp"
29              android:layout_marginTop="8dp"
30              android:layout_marginBottom="8dp">
31
32              <TextView
33                  android:id="@+id/tv_character"
34                  android:layout_width="wrap_content"
35                  android:layout_height="wrap_content"
36                  android:text="Carlotta"
37                  android:textColor="#000000"
38                  android:textSize="20sp"
```

```
39                  android:textStyle="bold"
40                  app:layout_constraintEnd_toEndOf="parent"
41
    app:layout_constraintStart_toEndOf="@+id/img_character"
42                  app:layout_constraintTop_toTopOf="parent" />
43
44              <ImageView
45                  android:id="@+id/img_character"
46                  android:layout_width="90dp"
47                  android:layout_height="143dp"
48
    app:layout_constraintBottom_toBottomOf="parent"
49
    app:layout_constraintEnd_toStartOf="@+id/tv_deskripsi"
50                  app:layout_constraintHorizontal_bias="0.4"
51
    app:layout_constraintStart_toStartOf="parent"
52                  app:layout_constraintTop_toTopOf="parent"
53                  app:layout_constraintVertical_bias="0.526"
54                  app:srcCompat="@drawable/card_carlotta" />
55
56              <TextView
57                  android:id="@+id/tv_deskripsi"
58                  android:layout_width="268dp"
59                  android:layout_height="62dp"
60                  android:layout_marginStart="11dp"
61                  android:text="Carlotta (Chinese: 珂莱塔) is a
    playable Glacio Mutant Resonator in Wuthering Waves. She is
    the second daughter of the esteemed Montelli family of
    Rinascita, embodying innate nobility and a refined
    appreciation for art."
62                  android:textColor="#000000"
63                  android:textSize="10sp"
64                  app:layout_constraintEnd_toEndOf="parent"
65
    app:layout_constraintStart_toEndOf="@+id/img_character"
66
    app:layout_constraintTop_toBottomOf="@+id/tv_character" />
67
68              <Button
69                  android:id="@+id/btn_detail"
70                  android:layout_width="wrap_content"
71                  android:layout_height="wrap_content"
72                  android:layout_marginStart="83dp"
73                  android:text="Detail"
74                  android:textColor="#000000"
                    app:cornerRadius="8dp"
75
76  app:layout_constraintBottom_toBottomOf="parent"

77  app:layout_constraintStart_toEndOf="@+id/img_character"
```

```
78
    app:layout_constraintTop_toBottomOf="@+id/tv_deskripsi"
79                  app:layout_constraintVertical_bias="1.0" />
80
81              <Button
82                  android:id="@+id/btn_link"
83                  android:layout_width="wrap_content"
84                  android:layout_height="wrap_content"
85                  android:layout_marginStart="20dp"
86                  android:layout_marginTop="12dp"
87                  android:layout_marginEnd="20dp"
88                  android:text="Link"
89                  android:textColor="#000000"
90                  app:cornerRadius="8dp"
91
    app:layout_constraintBottom_toBottomOf="parent"
92                  app:layout_constraintEnd_toEndOf="parent"
93
    app:layout_constraintStart_toEndOf="@+id/btn_detail"
94
    app:layout_constraintTop_toBottomOf="@+id/tv_deskripsi"
95                  app:layout_constraintVertical_bias="0.0" />
96          </androidx.constraintlayout.widget.ConstraintLayout>
97      </androidx.cardview.widget.CardView>
98  </androidx.constraintlayout.widget.ConstraintLayout>
```

10. HomeViewModel.kt

Tabel 10. Source Code HomeViewModel.kt

```
1   package com.example.wuwalist.presentation.home
2
3   import android.content.res.Resources
4   import androidx.lifecycle.ViewModel
5   import androidx.lifecycle.viewModelScope
6   import com.example.wuwalist.R
7   import com.example.wuwalist.model.Character
8   import kotlinx.coroutines.flow.Flow
9   import kotlinx.coroutines.flow.MutableStateFlow
10  import kotlinx.coroutines.flow.StateFlow
11  import kotlinx.coroutines.flow.flow
12  import kotlinx.coroutines.flow.onStart
13  import kotlinx.coroutines.launch
14
15  class HomeViewModel(private val resources: Resources) :
    ViewModel() {
16
17      private val _characterList =
    MutableStateFlow<List<Character>>(emptyList())
18      val characterList: StateFlow<List<Character>> get() =
    _characterList
19
20      private fun getCharacterFlow(): Flow<List<Character>> =
    flow {
21
22          val dataName =
    resources.getStringArray(R.array.nama_character)
23          val dataLink =
    resources.getStringArray(R.array.link_character)
24          val dataDescription =
    resources.getStringArray(R.array.deskripsi_character)
25          val dataPhoto =
    resources.obtainTypedArray(R.array.photo_character)
26
27          val listCharacter = ArrayList<Character>()
28          for (i in dataName.indices) {
29              val chara = Character(dataName[i], dataLink[i],
    dataDescription[i],dataPhoto.getResourceId(i, -1))
30              listCharacter.add(chara)
31          }
32          dataPhoto.recycle()
33          emit(listCharacter)
34      }
35
36      fun loadCharacters() {
37          viewModelScope.launch {
```

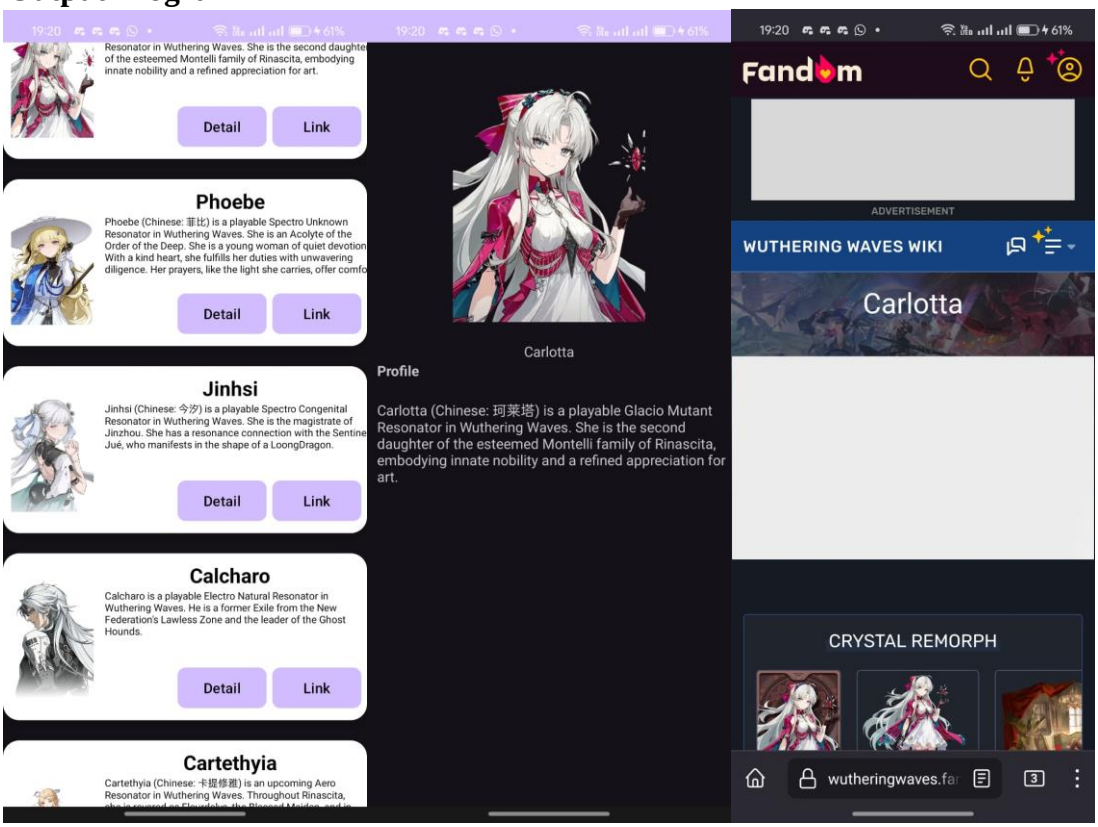
```
38              getCharacterFlow()
39                  .onStart {
40                      _characterList.value = emptyList()
41                  }
42                  .collect { characters ->
43                      _characterList.value = characters
44                  }
45          }
46      }
47  }
```

11. ViewModelFactory.kt

Tabel 11. Source Code ViewModelFactory.kt

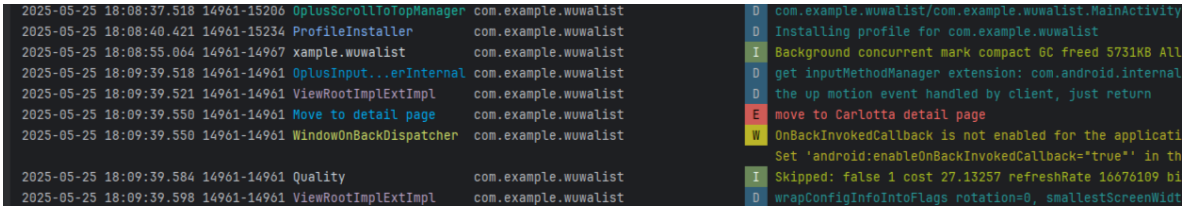
```
1   package com.example.wuwalist.utils
2
3   import android.content.res.Resources
4   import androidx.lifecycle.ViewModel
5   import androidx.lifecycle.ViewModelProvider
6   import com.example.wuwalist.presentation.home.HomeViewModel
7
8   class HomeViewModelFactory(private val resources: Resources)
    : ViewModelProvider.Factory {
9       override fun <T : ViewModel> create(modelClass: Class<T>):
    T {
10          if
    (modelClass.isAssignableFrom(HomeViewModel::class.java)) {
11              @Suppress("UNCHECKED_CAST")
12              return HomeViewModel(resources) as T
13          }
14          throw   IllegalArgumentException("Unknown   ViewModel
    class")
15      }
16  }
```

B. Output Program



Gambar 1. Screenshot Hasil Jawaban Soal 1

Gambar 2. Screenshot Log Saat Data Item Masuk Ke Dalam List



Gambar 3. Screenshot Log Saat Tombol Detail Dan Data Dari List Yang Dipilih Ketika Berpindah Ke Halaman Detail

Gambar 4. Screenshot Log Tombol Explicit Intent Ditekan

C. Pembahasan

1. MainActivity.kt:

- **Pada baris 1 hingga 9**, dilakukan impor berbagai library yang diperlukan, termasuk komponen dasar Android seperti Bundle, AppCompatActivity, dan utilitas untuk mengelola fragment. Impor ini memungkinkan aplikasi untuk menggunakan fitur-fitur Android modern seperti Fragment Manager dan logging.
- **Pada baris 11**, didefinisikan kelas MainActivity yang mewarisi AppCompatActivity, yang merupakan kelas dasar untuk aktivitas yang mendukung fitur-fitur kompatibilitas Android.
- **Pada baris 12 hingga 14**, terdapat method onCreate() yang merupakan entry point utama aktivitas. Di sini, layout activity_main.xml diinisialisasi menggunakan setContentView() untuk menampilkan antarmuka pengguna.
- **Pada baris 16 hingga 17**, dilakukan inisialisasi FragmentManager dan pembuatan instance HomeFragment. FragmentManager digunakan untuk mengelola fragment-fragment dalam aplikasi.
- **Pada baris 18**, dilakukan pengecekan apakah HomeFragment sudah ada dalam container menggunakan findFragmentByTag(). Ini mencegah duplikasi fragment saat aktivitas di-recreate, misalnya saat rotasi layar.
- **Pada baris 19 hingga 24**, terdapat blok kondisional yang hanya dijalankan jika HomeFragment belum ada. Di dalamnya, log debugging ditambahkan dan transaction fragment dilakukan untuk menambahkan HomeFragment ke container dengan ID frame_container.

2. Character.kt:

- **Pada baris 1 hingga 4**, dilakukan impor library yang diperlukan untuk membuat kelas data yang dapat di-parcel, yaitu Parcelable dan anotasi Parcelize.
- **Pada baris 6 hingga 12**, didefinisikan data class Character dengan anotasi @Parcelize yang mengimplementasikan interface Parcelable. Kelas ini memiliki empat properti: name (nama karakter), link (URL wiki), description (deskripsi karakter), dan photo (ID resource gambar).
- Penggunaan anotasi @Parcelize **pada baris 6** memungkinkan Kotlin untuk secara otomatis menghasilkan implementasi Parcelable tanpa perlu menulis kode boilerplate, sehingga memudahkan pengiriman objek Character antar komponen Android.

3. **ListCharacterAdapter.kt:**

- Pada **baris 1 hingga 9**, dilakukan impor berbagai library yang diperlukan untuk membuat adapter RecyclerView, termasuk komponen View, ViewGroup, dan widget-widget Android seperti Button, ImageView, dan TextView.
- Pada **baris 11 hingga 15**, didefinisikan kelas ListCharacterAdapter yang mewarisi RecyclerView.Adapter dengan tipe parameter ListViewHolder. Adapter ini menerima tiga parameter: listCharacter (sebuah ArrayList dari objek Character), onWikiClick (sebuah fungsi lambda yang akan dipanggil ketika tombol wiki diklik, membawa String sebagai argumen), dan onDetailClick (sebuah fungsi lambda yang akan dipanggil ketika tombol detail diklik, membawa String, Int, dan String sebagai argumen).
- Pada **baris 17 hingga 24**, didefinisikan inner class ListViewHolder yang mewarisi RecyclerView.ViewHolder. Kelas ini bertugas untuk menyimpan referensi ke view yang ada di dalam setiap item RecyclerView. Inisialisasi view (imgPhoto, tvName, tvDeskripsi, btnWiki, btnDetail) dilakukan menggunakan itemView.findViewById().
- Pada **baris 26 hingga 29**, diimplementasikan method onCreateViewHolder(). Method ini dipanggil ketika RecyclerView membutuhkan ViewHolder baru. Di sini, layout R.layout.item_character di-inflate (diubah dari XML menjadi objek View) menggunakan LayoutInflater dan kemudian sebuah instance ListViewHolder baru dibuat dan dikembalikan.
- Pada **baris 31**, diimplementasikan method getItemCount() yang mengembalikan jumlah total item dalam listCharacter.
- Pada **baris 33 hingga 40**, diimplementasikan method onBindViewHolder(). Method ini dipanggil oleh RecyclerView untuk menampilkan data pada posisi tertentu.
    - Pada **baris 34**, dilakukan destructuring declaration untuk mengambil properti name, link, description, dan photo dari objek Character pada posisi saat ini di listCharacter.
    - Pada **baris 35 hingga 37**, data karakter (nama, gambar, dan deskripsi) diatur ke TextView dan ImageView yang sesuai di dalam ViewHolder.
    - Pada **baris 38**, sebuah OnClickListener diatur untuk btnWiki. Ketika tombol ini diklik, fungsi lambda onWikiClick akan dipanggil dengan meneruskan link karakter.
- Pada **baris 39**, sebuah OnClickListener diatur untuk btnDetail. Ketika tombol ini diklik, fungsi lambda onDetailClick akan dipanggil dengan meneruskan name, photo, dan description karakter.
- Pada **baris 41 hingga 44**, didefinisikan fungsi setData() yang digunakan untuk memperbarui data di dalam adapter. Fungsi ini akan menghapus data lama dari listCharacter dan menambahkan semua data baru dari newList.

4. **HomeFragment.kt:**

- Pada **baris 1 hingga 16**, dilakukan impor library yang dibutuhkan, termasuk Fragment, Intent, Uri untuk intent eksplisit, komponen RecyclerView, ViewModel, LifecycleScope untuk coroutines, serta kelas-kelas dari paket proyek seperti ListCharacterAdapter, FragmentHomeBinding, HomeViewModelFactory, dan DetailFragment.
- Pada **baris 18 dan 19**, dideklarasikan variabel _binding dan binding untuk view binding dengan FragmentHomeBinding. Penggunaan !! (not-null asserted call) pada binding mengasumsikan _binding akan selalu terinisialisasi setelah onCreateView dan sebelum onDestroyView.
- Pada **baris 22**, dideklarasikan variabel characterAdapter untuk ListCharacterAdapter.
- Pada **baris 24 hingga 26**, viewModel diinisialisasi menggunakan delegasi viewModels dengan HomeViewModelFactory yang menerima resources sebagai dependensi. Ini memastikan HomeViewModel dibuat dengan benar.
- Pada **baris 28 hingga 33**, method onCreateView() meng-inflate layout FragmentHomeBinding dan mengembalikan binding.root sebagai view untuk fragment.
- Pada **baris 35 hingga 40**, method onViewCreated() dipanggil setelah view untuk fragment dibuat. Di sini, setupRecyclerView() dipanggil untuk menginisialisasi RecyclerView, observeCharacterList() dipanggil untuk mulai mengamati data dari ViewModel, dan viewModel.loadCharacters() dipanggil untuk memuat data karakter.

- Pada **baris 42 hingga 71**, didefinisikan fungsi setupRecyclerView():
    - Pada **baris 43 hingga 65**, characterAdapter diinisialisasi.
        - Parameter ArrayList() dikirim sebagai daftar awal (kosong) untuk adapter.
        - Parameter onWikiClick diisi dengan lambda yang akan membuat Intent dengan ACTION_VIEW untuk membuka link di browser. Terdapat Log.e untuk mencatat event saat tombol ini ditekan.
        - Parameter onDetailClick diisi dengan lambda yang akan membuat instance DetailFragment, menaruh data karakter (name, photo, description) ke dalam Bundle sebagai arguments, dan kemudian melakukan transaksi fragment untuk mengganti konten R.id.frame_container dengan detailFragment serta menambahkannya ke back stack. Terdapat Log.e untuk mencatat event saat tombol ini ditekan dan data yang dikirim.
    - Pada **baris 67 hingga 71**, RecyclerView (binding.rvCharacter) dikonfigurasi dengan LinearLayoutManager, characterAdapter yang sudah dibuat, dan setHasFixedSize(true) untuk optimasi karena ukuran item tidak berubah.
- Pada **baris 73 hingga 78**, didefinisikan fungsi observeCharacterList() yang menggunakan viewLifecycleOwner.lifecycleScope.launch untuk menjalankan coroutine yang akan mengamati viewModel.characterList (sebuah StateFlow). Ketika ada data baru, characterAdapter.setData(list) dipanggil untuk memperbarui RecyclerView. collectLatest digunakan untuk memastikan hanya data terbaru yang diproses.
- Pada **baris 80 hingga 83**, method onDestroyView() membersihkan _binding menjadi null untuk menghindari memory leaks.

5. DetailFragment.kt:

- **Pada baris 1 hingga 7**, dilakukan impor library yang diperlukan untuk membuat fragment dan menggunakan View Binding.
- **Pada baris 9 hingga 13**, didefinisikan kelas DetailFragment yang mewarisi Fragment. Di dalamnya, terdapat deklarasi variabel binding dengan pattern yang aman untuk memory leak.
- **Pada baris 15 hingga 32**, diimplementasikan method onCreateView() yang meng-inflate layout fragment_detail.xml menggunakan View Binding. **Pada baris 20-22**, data karakter diambil dari arguments. **Pada baris 24-28**, data tersebut ditampilkan pada view-view yang sesuai.
- **Pada baris 34 hingga 37**, diimplementasikan method onDestroyView() yang membersihkan referensi binding untuk mencegah memory leak.

6. activity_main.xml:

- **Pada baris 1 hingga 9**, didefinisikan layout utama aplikasi menggunakan FrameLayout. Layout ini diberi ID frame_container **pada baris 8**, yang digunakan sebagai container untuk fragment-fragment dalam aplikasi.

7. **fragment_home.xml:**

- **Pada baris 1 hingga 8**, didefinisikan layout untuk HomeFragment menggunakan ConstraintLayout. Layout ini dikonfigurasi dengan lebar dan tinggi wrap_content, yang mungkin perlu diubah menjadi match_parent untuk mengisi seluruh layar.
- **Pada baris 10 hingga 16**, didefinisikan RecyclerView dengan ID rv_character yang akan menampilkan daftar karakter. RecyclerView ini dikonfigurasi untuk mengisi seluruh parent layout menggunakan constraint.

8. **fragment_detail.xml:**

- **Pada baris 1 hingga 7**, didefinisikan layout untuk DetailFragment menggunakan ConstraintLayout dengan lebar dan tinggi match_parent.
- **Pada baris 9 hingga 16**, didefinisikan ImageView dengan ID img_character untuk menampilkan gambar karakter. ImageView ini dikonfigurasi dengan ukuran tetap dan diposisikan di tengah atas layout.
- **Pada baris 18 hingga 25**, didefinisikan TextView dengan ID tv_character untuk menampilkan nama karakter. TextView ini diposisikan di bawah gambar dan di tengah horizontal.
- **Pada baris 27 hingga 35**, didefinisikan TextView dengan ID tv_profile sebagai label untuk bagian profil. TextView ini diposisikan di bawah nama dengan alignment kiri dan style bold.
- **Pada baris 37 hingga 44**, didefinisikan TextView dengan ID tv_deskripsi untuk menampilkan deskripsi karakter. TextView ini diposisikan di bawah label profil dengan alignment kiri.

9. **item_character.xml:**

- **Pada baris 1 hingga 6**, didefinisikan layout root untuk item karakter menggunakan ConstraintLayout dengan lebar match_parent dan tinggi wrap_content.
- **Pada baris 8 hingga 24**, didefinisikan CardView dengan ID card_view yang memberikan efek card dengan sudut melengkung dan bayangan. CardView ini dikonfigurasi dengan berbagai properti seperti margin, warna latar, radius sudut, dan elevasi.
- **Pada baris 26 hingga 30**, didefinisikan ConstraintLayout di dalam CardView yang akan menampung semua elemen UI item karakter.
- **Pada baris 32 hingga 42**, didefinisikan TextView dengan ID tv_character untuk menampilkan nama karakter. TextView ini dikonfigurasi dengan style bold, ukuran 20sp, dan warna teks hitam.
- **Pada baris 44 hingga 55**, didefinisikan ImageView dengan ID img_character untuk menampilkan gambar karakter. ImageView ini dikonfigurasi dengan ukuran tetap dan diposisikan di sebelah kiri layout.
- **Pada baris 57 hingga 69**, didefinisikan TextView dengan ID tv_deskripsi untuk menampilkan deskripsi karakter. TextView ini dikonfigurasi dengan ukuran 10sp dan warna teks hitam.
- **Pada baris 71 hingga 82**, didefinisikan Button dengan ID btn_detail untuk melihat detail karakter. Button ini dikonfigurasi dengan sudut melengkung dan warna teks hitam.
- **Pada baris 84 hingga 96**, didefinisikan Button dengan ID btn_link untuk membuka link karakter di browser. Button ini juga dikonfigurasi dengan sudut melengkung dan warna teks hitam.

10. **HomeViewModel.kt:**

- Pada **baris 1 hingga 11**, dilakukan impor library yang diperlukan seperti Resources, ViewModel, viewModelScope untuk coroutines, kelas Character dari model, dan komponen Flow dari kotlinx.coroutines.flow.
- Pada **baris 13**, _characterList dideklarasikan sebagai MutableStateFlow yang privat, diinisialisasi dengan daftar kosong. Ini akan menyimpan state daftar karakter saat ini.
- Pada **baris 14**, characterList dideklarasikan sebagai StateFlow yang diekspos ke luar (bersifat read-only dari luar ViewModel). Ini adalah flow yang akan diamati oleh UI (dalam hal ini, HomeFragment).

- Pada **baris 16 hingga 29**, didefinisikan fungsi privat getCharacterFlow() yang mengembalikan Flow<List<Character>>. Fungsi ini bertugas untuk mengambil data karakter dari resources.
    - Pada **baris 18 hingga 21**, data nama, tautan, deskripsi, dan ID drawable foto karakter diambil dari arrays yang didefinisikan di strings.xml dan arrays.xml menggunakan resources.
    - Pada **baris 23 hingga 26**, sebuah ArrayList dari Character dibuat. Setiap karakter diinisialisasi dengan data yang telah diambil, dan kemudian ditambahkan ke listCharacter.
    - Pada **baris 27**, dataPhoto.recycle() dipanggil untuk melepaskan resources yang terkait dengan TypedArray setelah tidak lagi digunakan.
    - Pada baris 28, listCharacter di-emit melalui flow.
- Pada **baris 31 hingga 39**, didefinisikan fungsi loadCharacters(). Fungsi ini yang akan dipanggil dari HomeFragment untuk memicu pemuatan data.
    - viewModelScope.launch digunakan untuk menjalankan coroutine di dalam scope ViewModel.
    - getCharacterFlow() dipanggil untuk mendapatkan flow data.
    - .onStart { _characterList.value = emptyList() } digunakan untuk meng-emit daftar kosong ke _characterList segera sebelum flow mulai mengumpulkan data. Ini berguna untuk menampilkan state loading atau mengosongkan data sebelumnya.
    - .collect { characters -> _characterList.value = characters } mengumpulkan data yang di-emit dari getCharacterFlow() dan memperbarui _characterList.value dengan data karakter yang baru.

11. ViewModelFactory.kt:

- Pada **baris 1 hingga 5**, dilakukan impor library yang dibutuhkan, yaitu Resources, ViewModel, ViewModelProvider, dan HomeViewModel.
- Pada **baris 7**, kelas HomeViewModelFactory didefinisikan, yang menerima Resources sebagai parameter konstruktor. Kelas ini mengimplementasikan ViewModelProvider.Factory.
- Pada **baris 8 hingga 12**, method create() di-override. Method ini akan dipanggil oleh sistem ketika instance ViewModel dibutuhkan.
    - Pada **baris 9**, diperiksa apakah modelClass adalah turunan dari HomeViewModel::class.java.
    - Jika iya (pada **baris 11**), instance HomeViewModel dibuat dengan meneruskan resources yang diterima oleh factory, dan kemudian di-cast ke tipe T (dengan @Suppress("UNCHECKED_CAST") untuk menekan peringatan unchecked cast).
    - Jika tidak (pada **baris 14**), IllegalArgumentException dilempar karena factory ini hanya tahu cara membuat HomeViewModel.

12. Debugger:

Debugger adalah alat yang sangat penting dalam pengembangan perangkat lunak. Fungsinya adalah untuk membantu programmer menemukan dan memperbaiki bug atau kesalahan dalam kode aplikasi. Dengan debugger, Anda dapat menjalankan aplikasi secara terkontrol, baris per baris kode, memeriksa nilai variabel pada waktu tertentu, dan memahami alur eksekusi program. Ini memungkinkan Anda untuk mengidentifikasi mengapa aplikasi tidak berjalan sesuai harapan.

Cara Menggunakan Debugger

Berikut adalah langkah-langkah umum cara menggunakan Debugger di Android Studio:

1) **Menetapkan Breakpoint:**
   - Breakpoint adalah titik dalam kode di mana eksekusi program akan berhenti sementara, memungkinkan Anda untuk memeriksa state aplikasi.
   - Untuk menetapkan breakpoint, klik pada gutter (area di sebelah kiri nomor baris) di editor kode Android Studio pada baris yang ingin Anda selidiki. Sebuah titik merah akan muncul, menandakan breakpoint aktif. Anda perlu mencari setidaknya satu breakpoint yang relevan dengan aplikasi Anda.
2) **Memulai Sesi Debugging:**
   - Setelah breakpoint ditetapkan, jalankan aplikasi dalam mode debug. Anda bisa melakukannya dengan mengklik ikon "Debug 'app'" (biasanya bergambar serangga/bug) di toolbar Android Studio, atau melalui menu Run > Debug 'app'.
3) **Mengontrol Eksekusi:**
   - Ketika eksekusi program mencapai breakpoint, aplikasi akan berhenti, dan panel Debugger akan muncul di Android Studio. Panel ini menampilkan informasi seperti call stack (urutan pemanggilan fungsi), variabel lokal, dan nilai-nilainya.
   - Dari sini, Anda bisa menggunakan berbagai tombol kontrol untuk melanjutkan eksekusi.
4) **Memeriksa Variabel:**
   - Di panel Debugger, Anda dapat melihat nilai dari variabel-variabel yang ada dalam scope saat ini. Anda juga bisa "mengawasi" (watch) variabel tertentu untuk melihat perubahannya seiring eksekusi kode.

Fitur Step Into, Step Over, dan Step Out

Ketika eksekusi berhenti di sebuah breakpoint, Anda memiliki beberapa opsi untuk melanjutkan eksekusi baris per baris:

1) **Step Over:**
   - **Fungsi**: Mengeksekusi baris kode saat ini dan berpindah ke baris kode berikutnya dalam file yang sama.
   - **Kapan digunakan**: Jika baris saat ini berisi pemanggilan sebuah fungsi, "Step Over" akan mengeksekusi seluruh fungsi tersebut tanpa masuk ke dalamnya, lalu berhenti di baris berikutnya setelah pemanggilan fungsi itu selesai. Ini berguna jika Anda yakin fungsi tersebut bekerja dengan benar dan tidak perlu memeriksanya secara detail.

2) **Step Into:**
   - **Fungsi**: Jika baris kode saat ini adalah pemanggilan sebuah fungsi, "Step Into" akan masuk ke dalam fungsi tersebut, memungkinkan Anda untuk men-debug baris per baris di dalam fungsi itu.
   - **Kapan digunakan**: Gunakan ini jika Anda ingin memeriksa bagaimana sebuah fungsi tertentu dieksekusi atau jika Anda mencurigai ada bug di dalam fungsi tersebut. Jika baris saat ini bukan pemanggilan fungsi, "Step Into" akan berperilaku sama seperti "Step Over".
3) **Step Out:**
   - **Fungsi**: Mengeksekusi sisa baris kode dalam fungsi saat ini dan kemudian berhenti pada baris kode berikutnya setelah fungsi tersebut selesai dieksekusi (yaitu, kembali ke kode yang memanggil fungsi tersebut).
   - **Kapan digunakan**: Jika Anda sudah masuk ke dalam sebuah fungsi menggunakan "Step Into" dan telah selesai memeriksa apa yang Anda butuhkan di sana, Anda dapat menggunakan "Step Out" untuk keluar dari fungsi tersebut dengan cepat tanpa harus menjalankan sisa barisnya satu per satu.

D. Tautan Git

Berikut adalah tautan untuk source code yang telah dibuat.

https://github.com/PutraWhyra789/praktikum_pemrograman_mobile/tree/cdf23a7135dcda28e052a547837f2fe35d0abd1a/Module%204

SOAL 2

Jelaskan Application class dalam arsitektur aplikasi Android dan fungsinya

A. Jawaban

Dalam arsitektur aplikasi Android yang kompleks, Application class berperan sebagai komponen fundamental yang seringkali potensinya belum dimanfaatkan sepenuhnya. Ini adalah kelas dasar dalam sebuah aplikasi Android yang diinstansiasi sebelum komponen lain seperti Activity, Service, atau BroadcastReceiver ketika proses aplikasi dibuat. Anggaplah ini sebagai titik masuk paling awal untuk kode Anda, menyediakan sebuah singleton instance yang hidup selama proses aplikasi Anda berjalan. Memahami peran dan fungsinya adalah kunci untuk membangun aplikasi Android yang terstruktur dengan baik dan efisien.

Fungsi utama dari Application class adalah untuk memelihara state aplikasi global. Karena ia diinstansiasi pertama kali dan bertahan sepanjang siklus hidup aplikasi (kecuali prosesnya dihentikan), ini adalah tempat yang sesuai untuk menyimpan data atau objek yang perlu diakses secara luas di seluruh aplikasi. Ini bisa berkisar dari melakukan caching data bersama, mengelola sesi pengguna, atau menyimpan referensi ke kelas utilitas atau layanan yang dibutuhkan di berbagai bagian aplikasi.

Peran signifikan lainnya adalah inisialisasi global. Seringkali ada tugas atau library pihak ketiga yang perlu disiapkan sekali ketika aplikasi dimulai. Metode onCreate() dari Application class kustom adalah tempat yang sempurna untuk rutinitas inisialisasi sekali jalan tersebut. Ini bisa termasuk menyiapkan tools analitik, menginisialisasi framework logging, atau mengonfigurasi library jaringan. Dengan memusatkan inisialisasi ini di sini, Anda memastikan bahwa mereka hanya dilakukan sekali, mencegah panggilan setup yang berlebihan dan potensi konflik.

Lebih lanjut, Application class dapat merespons peristiwa siklus hidup tingkat aplikasi. Meskipun Activity memiliki siklus hidupnya sendiri yang detail, Application class juga memiliki callback seperti onCreate() dan onTerminate(). Meskipun onTerminate() tidak dijamin akan dipanggil pada perangkat produksi (ini terutama untuk lingkungan yang diemulasi), onCreate() dapat diandalkan untuk dipanggil dan merupakan kait utama untuk logika startup aplikasi. Ini juga dapat digunakan untuk bereaksi terhadap situasi memori rendah melalui onLowMemory() atau perubahan konfigurasi yang memengaruhi seluruh aplikasi melalui onConfigurationChanged(), meskipun penanganan yang terakhir lebih umum dilakukan di tingkat Activity.

Untuk memanfaatkan kemampuan ini, pengembang biasanya membuat subkelas kustom dari android.app.Application. Kelas kustom ini kemudian perlu dideklarasikan dalam file AndroidManifest.xml di dalam tag <application> menggunakan atribut android:name. Misalnya, jika Anda membuat kelas bernama MyGlobalApp, manifes Anda akan menyertakan <application android:name=".MyGlobalApp" ... >. Ini memberitahu sistem Android untuk menggunakan kelas kustom Anda sebagai titik masuk aplikasi.

Namun, sangat penting untuk menggunakan Application class dengan bijaksana. Membebaninya dengan terlalu banyak tanggung jawab atau menyimpan data dalam jumlah besar dapat menyebabkan desain monolitik dan peningkatan jejak memori. Ini juga dapat membuat pengujian menjadi lebih kompleks karena mengikat komponen dengan erat ke state global ini. Untuk mengelola dependensi dan state, pengembangan Android modern sering kali condong ke pola arsitektur seperti Dependency Injection (misalnya, Hilt atau Koin) yang dapat menawarkan solusi yang lebih skalabel dan dapat diuji daripada hanya mengandalkan Application class untuk semuanya.

Kesimpulannya, Application class adalah komponen yang kuat dalam framework Android, terutama dirancang untuk mengelola state aplikasi global dan melakukan inisialisasi sekali jalan. Ketika digunakan dengan cermat, ia menyediakan cara yang terpusat dan nyaman untuk menangani sumber daya dan konfigurasi yang menjangkau seluruh aplikasi. Namun, pengembang harus menyadari potensi kekurangannya dan menganggapnya sebagai salah satu alat di antara banyak alat lainnya dalam menyusun arsitektur aplikasi Android yang kuat dan dapat dipelihara.