package com.example.toolkit\_management\_mk3  
  
import android.content.Context  
import android.content.SharedPreferences  
import android.os.Bundle  
import android.widget.ArrayAdapter  
import android.widget.ListView  
import androidx.appcompat.app.AppCompatActivity  
import com.example.toolkit\_management\_mk3.models.Item  
import com.google.gson.Gson  
import com.google.gson.reflect.TypeToken  
  
class CheckActivity : AppCompatActivity() {  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_check*)  
  
 // 리스트뷰를 찾아옵니다.  
 val listViewRedItems: ListView = findViewById(R.id.*listViewRedItems*)  
  
 // 색상 상태를 로드합니다.  
 val colorPrefs = getSharedPreferences("colorPrefs", Context.*MODE\_PRIVATE*)  
 val colorState = loadColorState(colorPrefs)  
  
 // 모든 아이템을 로드하고 빨간색인 아이템만 필터링합니다.  
 val items = loadItems()  
 val redItems = items.*filter* **{** item **->** colorState.getOrDefault(item.id, 0) == 2 **}** // 어댑터를 설정하고 빨간색 항목들을 리스트뷰에 표시합니다.  
 val adapter = ArrayAdapter(this, android.R.layout.*simple\_list\_item\_1*, redItems.*map* **{ it**.mainText **}**)  
 listViewRedItems.*adapter* = adapter  
 }  
  
 private fun loadItems(): List<Item> {  
 val sharedPref = getSharedPreferences("AppPrefs", Context.*MODE\_PRIVATE*)  
 val json = sharedPref.getString("items", null)  
 return if (json != null) {  
 Gson().fromJson(json, object : TypeToken<List<Item>>() {}.*type*)  
 } else {  
 *listOf*()  
 }  
 }  
  
 private fun loadColorState(prefs: SharedPreferences): Map<Int, Int> {  
 val json = prefs.getString("colorState", null)  
 return if (json != null) {  
 Gson().fromJson(json, object : TypeToken<Map<Int, Int>>() {}.*type*)  
 } else {  
 *mapOf*()  
 }  
 }  
}

package com.example.toolkit\_management\_mk3  
  
import android.content.Context  
import android.content.SharedPreferences  
import android.graphics.Color  
import android.graphics.drawable.ColorDrawable  
import android.os.Bundle  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import android.widget.ArrayAdapter  
import android.widget.ListView  
import androidx.fragment.app.DialogFragment  
import com.example.toolkit\_management\_mk3.models.Item  
import com.google.gson.Gson  
import com.google.gson.reflect.TypeToken  
  
class CheckDialogFragment : DialogFragment() {  
 private lateinit var listView: ListView  
 private var itemList = *mutableListOf*<Item>()  
 private lateinit var adapter: ArrayAdapter<String>  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 // Set the style for the dialog here using the CustomDialogStyle defined in styles.xml  
 setStyle(DialogFragment.*STYLE\_NORMAL*, R.style.*CustomDialogStyle*)  
 }  
  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View? {  
 // Inflate the layout for this fragment  
 return inflater.inflate(R.layout.*fragment\_check\_dialog*, container, false)  
 }  
  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
 listView = view.findViewById(R.id.*listViewRedItems*)  
 // 여기에서 custom\_textview 레이아웃을 사용하도록 ArrayAdapter를 설정합니다.  
 adapter = ArrayAdapter(requireContext(), R.layout.*custom\_textview*, R.id.*customTextView*, *mutableListOf*<String>())  
 listView.*adapter* = adapter  
 loadData()  
 }  
  
 private fun loadData() {  
 val sharedPreferences = requireActivity().getSharedPreferences("colorPrefs", Context.*MODE\_PRIVATE*)  
 val colorState = loadColorState(sharedPreferences)  
  
 val sharedPrefItems = requireActivity().getSharedPreferences("AppPrefs", Context.*MODE\_PRIVATE*)  
 val json = sharedPrefItems.getString("items", null)  
 json?.*let* **{** val type = object : TypeToken<List<Item>>() {}.*type* itemList = Gson().fromJson(**it**, type)  
 val filteredItems = itemList.*filter* **{** item **->** colorState[item.id] == 2 **}** adapter.clear()  
 adapter.addAll(filteredItems.*map* **{ it**.mainText **}**)  
 adapter.notifyDataSetChanged()  
 **}** }  
  
 private fun loadColorState(prefs: SharedPreferences): Map<Int, Int> {  
 val json = prefs.getString("colorState", null)  
 return if (json != null) {  
 Gson().fromJson(json, object : TypeToken<Map<Int, Int>>() {}.*type*)  
 } else {  
 *mutableMapOf*()  
 }  
 }  
  
 override fun onStart() {  
 super.onStart()  
 *dialog*?.*window*?.*let* **{** window **->** val width = ViewGroup.LayoutParams.*MATCH\_PARENT* val height = ViewGroup.LayoutParams.*WRAP\_CONTENT* window.setLayout(width, height)  
 window.setBackgroundDrawable(ColorDrawable(Color.*TRANSPARENT*))  
 **}** }  
  
 companion object {  
 fun newInstance() = CheckDialogFragment()  
 }  
}

package com.example.toolkit\_management\_mk3  
  
import android.content.Context  
import android.util.Log  
import android.view.KeyEvent  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import android.view.inputmethod.EditorInfo  
import android.view.inputmethod.InputMethodManager  
import android.widget.ArrayAdapter  
import android.widget.Button  
import android.widget.EditText  
import com.example.toolkit\_management\_mk3.models.Item  
import com.google.gson.Gson  
  
class CustomItemAdapter\_1(  
 private val context: Context,  
 private val resource: Int,  
 private var items: MutableList<Item>  
) : ArrayAdapter<Item>(context, resource, items) {  
  
 private val sharedPreferences = context.getSharedPreferences("AppPrefs", Context.*MODE\_PRIVATE*)  
  
 override fun getView(position: Int, convertView: View?, parent: ViewGroup): View {  
 val holder: ViewHolder  
 val view: View  
  
 if (convertView == null) {  
 view = LayoutInflater.from(context).inflate(resource, parent, false)  
 holder = ViewHolder(view.findViewById(R.id.*editTextItem*), view.findViewById(R.id.*editTextSubItem*), view.findViewById(R.id.*buttonDelete*))  
 view.*tag* = holder  
 } else {  
 view = convertView  
 holder = view.*tag* as ViewHolder  
 }  
  
 holder.editTextItem.setText(items[position].mainText)  
 holder.editTextSubItem.setText(items[position].subItems)  
 holder.editTextItem.setSelection(items[position].mainText.length)  
  
 // EditorActionListener를 설정하여 'Done' 버튼 입력을 감지합니다.  
 holder.editTextItem.setOnEditorActionListener **{** v, actionId, event **->** if (actionId == EditorInfo.*IME\_ACTION\_DONE* || (event != null && event.*action* == KeyEvent.*ACTION\_UP* && event.*keyCode* == KeyEvent.*KEYCODE\_ENTER*)) {  
 saveData(position, holder.editTextItem.*text*.toString(), holder.editTextSubItem.*text*.toString())  
 val imm = context.getSystemService(Context.*INPUT\_METHOD\_SERVICE*) as InputMethodManager  
 imm.hideSoftInputFromWindow(v.*windowToken*, 0)  
 v.clearFocus()  
 true  
 } else {  
 false  
 }  
 **}** // Adding similar listeners for editTextSubItem  
 holder.editTextSubItem.setOnEditorActionListener **{** v, actionId, event **->** if (actionId == EditorInfo.*IME\_ACTION\_DONE* || (event != null && event.*action* == KeyEvent.*ACTION\_UP* && event.*keyCode* == KeyEvent.*KEYCODE\_ENTER*)) {  
 saveData(position, holder.editTextItem.*text*.toString(), holder.editTextSubItem.*text*.toString())  
 val imm = context.getSystemService(Context.*INPUT\_METHOD\_SERVICE*) as InputMethodManager  
 imm.hideSoftInputFromWindow(v.*windowToken*, 0)  
 v.clearFocus()  
 true  
 } else {  
 false  
 }  
 **}** // 포커스 리스너를 설정합니다.  
 holder.editTextItem.setOnFocusChangeListener **{** v, hasFocus **->** if (!hasFocus) {  
 saveData(position, holder.editTextItem.*text*.toString(), holder.editTextSubItem.*text*.toString())  
 }  
 **}** // editTextSubItem에 대한 포커스 리스너 추가  
 holder.editTextSubItem.setOnFocusChangeListener **{** v, hasFocus **->** if (!hasFocus) {  
 saveData(position, holder.editTextItem.*text*.toString(), holder.editTextSubItem.*text*.toString())  
 }  
 **}** holder.deleteButton.setOnClickListener **{** items.removeAt(position)  
 notifyDataSetChanged()  
 saveData()  
 Log.d("CustomItemAdapter\_1", "Item deleted at position: $position")  
 **}** return view  
 }  
  
 private fun saveData(position: Int, mainText: String, subText: String) {  
 val item = items[position]  
 if (item.mainText != mainText || item.subItems != subText) {  
 item.mainText = mainText  
 item.subItems = subText  
 notifyDataSetChanged()  
 val json = Gson().toJson(items)  
 sharedPreferences.edit().putString("items", json).apply()  
 Log.d("CustomItemAdapter\_1", "Data saved: $json")  
 }  
 }  
  
 private class ViewHolder(val editTextItem: EditText, val editTextSubItem: EditText, val deleteButton: Button)  
  
 fun updateItems(newItems: List<Item>) {  
 items.clear()  
 items.addAll(newItems)  
 notifyDataSetChanged() // 데이터 변경 알림 및 UI 갱신  
 saveData() // Update the SharedPreferences whenever the list is updated  
 }  
  
 fun saveData() {  
 val json = Gson().toJson(items)  
 sharedPreferences.edit().putString("items", json).apply()  
 Log.d("CustomItemAdapter\_1", "Data saved: $json")  
 }  
}

package com.example.toolkit\_management\_mk3  
  
import android.content.Context  
import android.graphics.Color  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import android.widget.ArrayAdapter  
import android.widget.TextView  
import com.example.toolkit\_management\_mk3.models.Item  
import com.google.gson.Gson  
  
class CustomItemAdapter\_2(  
 private val context: Context,  
 private val resource: Int,  
 private var items: MutableList<Item>,  
 private var colorState: MutableMap<Int, Int>  
) : ArrayAdapter<Item>(context, resource, items) {  
  
 private val sharedPreferences = context.getSharedPreferences("AppPrefs", Context.*MODE\_PRIVATE*)  
  
 override fun getView(position: Int, convertView: View?, parent: ViewGroup): View {  
 val itemView = convertView ?: LayoutInflater.from(context).inflate(resource, parent, false)  
 val textViewItem = itemView.findViewById<TextView>(R.id.*textViewItemMain*)  
 val subTextView = itemView.findViewById<TextView>(R.id.*textViewItemSub*)  
  
 val item = getItem(position) ?: return itemView // Handle null case for item  
 textViewItem.*text* = item.mainText  
 subTextView.*text* = item.subItems  
  
 // Set the color based on the stored state  
 val color = getColorForState(item.id)  
 textViewItem.setTextColor(color)  
 subTextView.setTextColor(color)  
  
 // Set up click listener to change the color  
 textViewItem.setOnClickListener **{** val currentState = colorState[item.id] ?: 0 // Default to 0 if not found  
 val newState = (currentState + 1) % 3 // Cycle through 0 (Black), 1 (Blue), 2 (Red)  
 colorState[item.id] = newState  
 saveColorState() // Save the updated color state  
 notifyDataSetChanged() // Refresh the ListView to display changes  
 **}** return itemView  
 }  
  
 private fun getColorForState(itemId: Int): Int {  
 // Determine the color based on the state  
 return when (colorState.getOrDefault(itemId, 0)) {  
 1 -> Color.*BLUE* 2 -> Color.*RED* else -> Color.*BLACK* }  
 }  
  
 private fun saveColorState() {  
 // Save the color state to SharedPreferences  
 val json = Gson().toJson(colorState)  
 sharedPreferences.edit().putString("colorState", json).apply()  
 }  
  
 fun updateItems(newItems: MutableList<Item>) {  
 items = newItems  
 notifyDataSetChanged() // Notify the adapter to refresh the list view  
 }  
}

package com.example.toolkit\_management\_mk3.com.example.toolkit\_management\_mk3  
  
import android.content.Context  
import android.graphics.Color  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import android.widget.ArrayAdapter  
import android.widget.TextView  
import com.example.toolkit\_management\_mk3.R  
import com.example.toolkit\_management\_mk3.models.Item  
  
class DDFAdapter(context: Context, private var items: MutableList<Item>, private val colorState: MutableMap<Int, Int>)  
 : ArrayAdapter<Item>(context, 0, items) {  
  
 override fun getView(position: Int, convertView: View?, parent: ViewGroup): View {  
 val itemView = convertView ?: LayoutInflater.from(*context*).inflate(R.layout.*custom\_dispactview*, parent, false)  
 val holder = itemView.*tag* as? ViewHolder ?: ViewHolder(itemView).*also* **{** itemView.*tag* = **it }** val item = getItem(position) ?: return itemView // null check 추가  
 holder.mainTextView.*text* = item.mainText  
 holder.subTextView.*text* = item.subItems  
  
 val color = getColorForState(item.id)  
 holder.mainTextView.setTextColor(color)  
 holder.subTextView.setTextColor(color)  
  
 return itemView  
 }  
  
 private fun getColorForState(itemId: Int): Int {  
 return when (colorState[itemId] ?: 0) {  
 1 -> Color.*BLUE* 2 -> Color.*RED* else -> Color.*BLACK* }  
 }  
  
 class ViewHolder(view: View) {  
 val mainTextView: TextView = view.findViewById(R.id.*mainTextView*)  
 val subTextView: TextView = view.findViewById(R.id.*subTextView*)  
 }  
  
 fun updateItems(newItems: List<Item>) {  
 clear()  
 addAll(newItems)  
 notifyDataSetChanged()  
 }  
}

package com.example.toolkit\_management\_mk3  
  
import android.content.Context  
import android.os.Bundle  
import android.util.Log  
import android.widget.Button  
import android.widget.ListView  
import androidx.appcompat.app.AppCompatActivity  
import androidx.lifecycle.Observer  
import androidx.lifecycle.ViewModelProvider  
import com.example.toolkit\_management\_mk3.com.example.toolkit\_management\_mk3.ColorViewModel  
import com.example.toolkit\_management\_mk3.models.Item  
import com.google.gson.Gson  
import com.google.gson.reflect.TypeToken  
  
class DisplayActivity : AppCompatActivity() {  
 private lateinit var listView: ListView  
 private var adapter: CustomItemAdapter\_2? = null  
 private var itemList = *mutableListOf*<Item>()  
 private lateinit var viewModel: ColorViewModel  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_display*)  
  
 listView = findViewById(R.id.*listViewItems*)  
 viewModel = ViewModelProvider(this).get(ColorViewModel::class.*java*)  
 viewModel.loadColorState(this)  
  
 loadData() // 어댑터 초기화와 데이터 로드  
  
 viewModel.colorState.observe(this, *Observer* **{** updatedColorState **->** adapter?.updateColorState(updatedColorState) // null-safe call 사용  
 adapter?.notifyDataSetChanged() // 색상 변경에 대한 UI 업데이트  
 **}**)  
  
 val buttonEdit: Button = findViewById(R.id.*buttonEdit*)  
 buttonEdit.setOnClickListener **{** showDisplayDialog()  
 **}** }  
  
 override fun onResume() {  
 super.onResume()  
 loadData() // onResume에서도 데이터 로드  
 }  
  
 private fun loadData() {  
 val sharedPreferences = getSharedPreferences("AppPrefs", Context.*MODE\_PRIVATE*)  
 val itemsJson = sharedPreferences.getString("items", null)  
 if (itemsJson != null) {  
 itemList = Gson().fromJson(itemsJson, object : TypeToken<List<Item>>() {}.*type*)  
 if (adapter == null) {  
 adapter = CustomItemAdapter\_2(this, R.layout.*display\_item\_layout*, itemList, *mutableMapOf*())  
 listView.*adapter* = adapter // 어댑터를 리스트뷰에 설정  
 } else {  
 adapter?.updateItems(itemList) // 기존 어댑터에 아이템 업데이트  
 }  
 adapter?.notifyDataSetChanged() // UI 갱신  
 } else {  
 Log.d("DisplayActivity", "No items found in SharedPreferences.")  
 }  
 }  
  
 fun showDisplayDialog() {  
 val fragment = DisplayDialogFragment.newInstance()  
 fragment.show(*supportFragmentManager*, "displayDialog")  
 }  
}

package com.example.toolkit\_management\_mk3  
  
import android.content.Context  
import android.content.Intent  
import android.graphics.Color  
import android.graphics.drawable.ColorDrawable  
import android.os.Bundle  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import android.widget.Button  
import android.widget.ListView  
import androidx.fragment.app.DialogFragment  
import androidx.lifecycle.ViewModelProvider  
import com.example.toolkit\_management\_mk3.com.example.toolkit\_management\_mk3.ColorViewModel  
import com.example.toolkit\_management\_mk3.com.example.toolkit\_management\_mk3.DDFAdapter  
import com.example.toolkit\_management\_mk3.models.Item  
import com.google.gson.Gson  
import com.google.gson.reflect.TypeToken  
  
class DisplayDialogFragment : DialogFragment() {  
 private lateinit var listView: ListView  
 private var adapter: DDFAdapter? = null  
 private var itemList: MutableList<Item> = *mutableListOf*()  
 private var colorState: MutableMap<Int, Int> = *mutableMapOf*()  
 private lateinit var viewModel: ColorViewModel  
  
 companion object {  
 fun newInstance() = DisplayDialogFragment()  
 }  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setStyle(DialogFragment.*STYLE\_NORMAL*, R.style.*CustomDialogStyle*)  
  
 // ViewModel 초기화  
 viewModel = ViewModelProvider(requireActivity()).get(ColorViewModel::class.*java*)  
 }  
  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?  
 ): View? {  
 return inflater.inflate(R.layout.*fragment\_display\_dialog*, container, false)  
 }  
  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
 listView = view.findViewById(R.id.*listViewDisplayItems*)  
 loadData()  
 adapter = DDFAdapter(requireContext(), itemList, colorState)  
 listView.*adapter* = adapter  
  
 // 색상 상태 관찰  
 viewModel.colorState.observe(*viewLifecycleOwner*) **{** updatedColorState **->** colorState = updatedColorState  
 adapter?.updateColorState(updatedColorState)  
 adapter?.notifyDataSetChanged()  
 **}** setupItemListeners()  
  
 val closeButton: Button = view.findViewById(R.id.*buttonClose*)  
 closeButton.setOnClickListener **{** dismiss()  
 **}** val editButton: Button = view.findViewById(R.id.*buttonEdit*)  
 editButton.setOnClickListener **{** startActivity(Intent(*activity*, PocketActivity::class.*java*))  
 **}** }  
  
 private fun setupItemListeners() {  
 listView.setOnItemClickListener **{** \_, \_, position, \_ **->** val item = itemList.*getOrNull*(position)  
 item?.let **{** val currentState = colorState[**it**.id] ?: 0  
 val newState = (currentState + 1) % 3  
 colorState[**it**.id] = newState  
 viewModel.updateColorState(requireContext(), **it**.id, newState)  
 adapter?.notifyDataSetChanged()  
 **}  
 }** }  
  
 private fun loadData() {  
 val sharedPreferences = requireActivity().getSharedPreferences("AppPrefs", Context.*MODE\_PRIVATE*)  
 val itemsJson = sharedPreferences.getString("items", null)  
 val colorStateJson = sharedPreferences.getString("colorState", "{}")  
 itemList = Gson().fromJson(itemsJson, object : TypeToken<MutableList<Item>>() {}.*type*)  
 colorState = Gson().fromJson(colorStateJson, object : TypeToken<MutableMap<Int, Int>>() {}.*type*)  
 adapter?.updateItems(itemList)  
 }  
  
 override fun onStart() {  
 super.onStart()  
 *dialog*?.*window*?.*apply* **{** setLayout(ViewGroup.LayoutParams.*MATCH\_PARENT*, ViewGroup.LayoutParams.*WRAP\_CONTENT*)  
 setBackgroundDrawable(ColorDrawable(Color.*TRANSPARENT*))  
 **}** }  
}

package com.example.toolkit\_management\_mk3.com.example.toolkit\_management\_mk3  
  
import android.content.Context  
import android.util.AttributeSet  
import android.view.MotionEvent  
import android.widget.LinearLayout  
import android.view.inputmethod.InputMethodManager  
import android.app.Activity  
  
class FocusHandlingLayout : LinearLayout {  
 constructor(context: Context) : super(context)  
 constructor(context: Context, attrs: AttributeSet?) : super(context, attrs)  
 constructor(context: Context, attrs: AttributeSet?, defStyleAttr: Int) : super(context, attrs, defStyleAttr)  
  
 init {  
 isFocusable = true  
 isClickable = true  
 }  
  
 override fun performClick(): Boolean {  
 super.performClick()  
 return true  
 }  
  
 override fun onInterceptTouchEvent(ev: MotionEvent?): Boolean {  
 if (ev?.action == MotionEvent.ACTION\_DOWN) {  
 val activity = context as? Activity  
 val currentFocusedView = activity?.currentFocus  
 if (currentFocusedView != null) {  
 val imm = context.getSystemService(Context.INPUT\_METHOD\_SERVICE) as InputMethodManager  
 imm.hideSoftInputFromWindow(currentFocusedView.windowToken, 0)  
 currentFocusedView.clearFocus()  
 }  
 }  
 return super.onInterceptTouchEvent(ev)  
 }  
}

package com.example.toolkit\_management\_mk3  
  
import android.os.Bundle  
import android.widget.Button  
import android.widget.TextView  
import androidx.appcompat.app.AppCompatActivity  
import com.example.toolkit\_management\_mk3.models.Item  
import com.google.gson.Gson  
  
class MainActivity : AppCompatActivity() {  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_main*)  
  
 val pocketA: TextView = findViewById(R.id.*pocketA*)  
 pocketA.setOnClickListener **{** // DisplayDialogFragment를 생성  
 val displayFragment = DisplayDialogFragment.newInstance()  
 displayFragment.show(*supportFragmentManager*, "displayDialog")  
 **}** val buttonCheck: Button = findViewById(R.id.*buttonCheck*)  
 buttonCheck.setOnClickListener **{** // CheckDialogFragment를 표시  
 val dialog = CheckDialogFragment.newInstance()  
 dialog.show(*supportFragmentManager*, "checkDialog")  
 **}** }  
  
 private fun saveItemsToPreferences(items: ArrayList<Item>) {  
 val sharedPreferences = getSharedPreferences("AppPrefs", *MODE\_PRIVATE*)  
 val editor = sharedPreferences.edit()  
 val json = Gson().toJson(items)  
 editor.putString("items", json)  
 editor.apply()  
 }  
}

package com.example.toolkit\_management\_mk3  
  
import android.annotation.SuppressLint  
import android.content.Context  
import android.os.Bundle  
import android.view.inputmethod.InputMethodManager  
import android.widget.Button  
import android.widget.EditText  
import android.widget.ListView  
import androidx.appcompat.app.AppCompatActivity  
import com.example.toolkit\_management\_mk3.models.Item  
import com.google.gson.Gson  
import com.google.gson.reflect.TypeToken  
  
class PocketActivity : AppCompatActivity() {  
 private lateinit var listView: ListView  
 private lateinit var adapter: CustomItemAdapter\_1  
 private var itemList = *mutableListOf*<Item>()  
 private lateinit var editTextMainItem: EditText  
 private lateinit var editTextSubItem: EditText  
 private lateinit var buttonAddItem: Button  
 private var isInitialLoad = true  
  
 @SuppressLint("ClickableViewAccessibility")  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activitypocket*)  
  
 listView = findViewById(R.id.*listViewItems*)  
 editTextMainItem = findViewById(R.id.*editTextNewItem*)  
 editTextSubItem = findViewById(R.id.*editTextSubItem*) // Initializing the sub item EditText  
 buttonAddItem = findViewById(R.id.*buttonAddItem*)  
 adapter = CustomItemAdapter\_1(this, R.layout.*item\_layout*, itemList)  
 listView.*adapter* = adapter  
  
 loadData()  
  
 // Hide keyboard and clear focus when touching outside EditText fields  
 listView.setOnTouchListener **{** \_, event **->** if (*currentFocus* != null) {  
 val imm = getSystemService(Context.*INPUT\_METHOD\_SERVICE*) as InputMethodManager  
 imm.hideSoftInputFromWindow(*currentFocus*?.*windowToken*, 0)  
 *currentFocus*?.clearFocus()  
 }  
 true // 이벤트 처리됨을 나타냄  
 **}** buttonAddItem.setOnClickListener **{** val mainItemText = editTextMainItem.*text*.toString()  
 val subItemText = editTextSubItem.*text*.toString()  
  
 if (mainItemText.*isNotBlank*()) {  
 val prefs = getSharedPreferences("AppPrefs", Context.*MODE\_PRIVATE*)  
 val lastId = prefs.getInt("lastItemId", 0) + 1  
 prefs.edit().putInt("lastItemId", lastId).apply()  
  
 val newItem = Item(lastId, mainItemText, subItemText)  
 loadData()  
 itemList.add(newItem)  
 adapter.updateItems(itemList)  
 editTextMainItem.*text*.clear()  
 editTextSubItem.*text*.clear()  
 adapter.saveData()  
 }  
 **}** }  
  
 private fun loadData() {  
 val sharedPref = getSharedPreferences("PocketPrefs", Context.*MODE\_PRIVATE*)  
 val json = sharedPref.getString("items", null)  
 itemList = if (json != null) {  
 Gson().fromJson(json, object : TypeToken<MutableList<Item>>() {}.*type*)  
 } else {  
 *mutableListOf*()  
 }  
 adapter.updateItems(itemList)  
 }  
  
 override fun onResume() {  
 super.onResume()  
 if (!isInitialLoad) {  
 loadData() // Reload data only if returning from another activity  
 }  
 isInitialLoad = false  
 }  
  
 override fun onPause() {  
 super.onPause()  
 adapter.saveData() // Save data when the activity is paused  
 }  
}

package com.example.toolkit\_management\_mk3.com.example.toolkit\_management\_mk3  
  
import android.content.Context  
import androidx.lifecycle.MutableLiveData  
import androidx.lifecycle.ViewModel  
import com.google.gson.Gson  
import com.google.gson.reflect.TypeToken  
  
class ColorViewModel : ViewModel() {  
 val colorState: MutableLiveData<MutableMap<Int, Int>> = MutableLiveData()  
  
 fun loadColorState(context: Context) {  
 val sharedPreferences = context.getSharedPreferences("AppPrefs", Context.*MODE\_PRIVATE*)  
 val colorStateJson = sharedPreferences.getString("colorState", "{}")  
 val type = object : TypeToken<MutableMap<Int, Int>>() {}.*type* colorState.*value* = Gson().fromJson(colorStateJson, type)  
 }  
  
 fun updateColorState(context: Context, itemId: Int, newState: Int) {  
 val sharedPreferences = context.getSharedPreferences("AppPrefs", Context.*MODE\_PRIVATE*)  
 val currentColorState = colorState.*value* ?: *mutableMapOf*()  
 currentColorState[itemId] = newState  
  
 colorState.*value* = currentColorState // Update LiveData  
  
 *with*(sharedPreferences.edit()) **{** putString("colorState", Gson().toJson(currentColorState))  
 apply()  
 **}** }  
}

package com.example.toolkit\_management\_mk3.models  
  
import android.graphics.Color  
import android.os.Parcelable  
import kotlinx.parcelize.Parcelize  
  
@Parcelize  
data class Item(  
 val id: Int,  
 var mainText: String,  
 var subItems: String,  
 var color: Int = Color.*BLACK*) : Parcelable

app

AndroidManifest.xml

toolkit\_management\_mk3

theme

CheckActivity

Color.kt

Theme.kt

Type.kt

CheckDialogFragment

CustomItemAdapter\_1

CustomItemAdapter\_2

DDFAdapter

DisplayActivity

DisplayDialogFragment

FocusHandlingLayout

MainActivity

PocketActivity

ViewModel.kt

toolkit\_management\_mk3

toolkit\_management\_mk3

models

Item

CustomItemAdapter

drawable

layout

activity\_check.xml

activity\_display.xml

activity\_main.xml

activitypocket.xml

custom\_dispactview.xml

custom\_textview.xml

display\_item\_layout.xml

fragment\_check\_dialog.xml

fragment\_display\_dialog.xml

item\_layout.xml

mipmap-hdpi

mipmap-mdpi

mipmap-xhdpi

mipmap-anydpi

mipmap-xxhdpi

mipmap-xxxhdpi

values

colors.xml

strings.xml

styles.xml

themes.xml

xml

backup\_rules.xml

data\_extraction\_rules.xml

build.gradle.kts

build.gradle.kts

proguard-rules.pro

gradle.properties

gradle-wrapper.properties

libs.versions.toml

local.properties

settings.gradle.kts

<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools">  
  
 <application  
 android:allowBackup="true"  
 android:dataExtractionRules="@xml/data\_extraction\_rules"  
 android:fullBackupContent="@xml/backup\_rules"  
 android:icon="@mipmap/ic\_launcher"  
 android:label="@string/app\_name"  
 android:roundIcon="@mipmap/ic\_launcher\_round"  
 android:supportsRtl="true"  
 android:theme="@style/AppTheme"  
 tools:targetApi="31">  
 <activity  
 android:name=".CheckDialogFragment"  
 android:hardwareAccelerated="true"  
 android:theme="@style/CustomDialogStyle"/>  
 <activity  
 android:name=".DisplayActivity"  
 android:label="@string/title\_activity\_display"  
 android:theme="@style/AppTheme"  
 android:exported="true"/>  
 <activity  
 android:name=".PocketActivity"  
 android:exported="false"  
 android:label="@string/title\_activity\_pocket"  
 android:theme="@style/AppTheme"  
 android:windowSoftInputMode="adjustResize" />  
 <activity  
 android:name=".CheckActivity"  
 android:label="@string/title\_activity\_check"  
 android:theme="@style/AppTheme" />  
 <activity  
 android:name=".MainActivity"  
 android:exported="true"  
 android:label="@string/app\_name"  
 android:theme="@style/AppTheme">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN" />  
 <category android:name="android.intent.category.LAUNCHER" />  
 </intent-filter>  
 </activity>  
 </application>  
</manifest>

import com.android.build.api.dsl.Packaging  
  
plugins **{** id("com.android.application")  
 id("kotlin-android")  
 id("org.jetbrains.kotlin.plugin.parcelize")  
**}**android **{** namespace = "com.example.toolkit\_management\_mk3"  
 compileSdk = 34  
  
 defaultConfig **{** applicationId = "com.example.toolkit\_management\_mk3"  
 minSdk = 34  
 targetSdk = 34  
 versionCode = 1  
 versionName = "1.0"  
  
 testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"  
 **}** buildTypes **{** release **{** isMinifyEnabled = false  
 proguardFiles(getDefaultProguardFile("proguard-android-optimize.txt"), "proguard-rules.pro")  
 **}  
 }** buildFeatures **{** compose = true  
 **}** composeOptions **{** kotlinCompilerExtensionVersion = "1.5.1"  
 **}** fun Packaging.() {  
 resources **{** excludes.add("/META-INF/{AL2.0,LGPL2.1}")  
 **}** }  
**}**dependencies **{** implementation("androidx.core:core-ktx:1.12.0")  
 implementation("androidx.appcompat:appcompat:1.6.1")  
 implementation("com.google.android.material:material:1.11.0")  
 implementation("com.google.code.gson:gson:2.10.1")  
  
// Compose UI Components  
 implementation("androidx.compose.ui:ui:1.5.1")  
 implementation("androidx.compose.ui:ui-graphics:1.5.1")  
 implementation("androidx.compose.ui:ui-tooling-preview:1.5.1")  
 implementation("androidx.activity:activity-compose:1.5.1")  
 implementation("androidx.compose.material3:material3:1.2.0")  
 implementation("androidx.compose.foundation:foundation-layout:1.5.1")  
  
//parcelize 관련  
  
//colorviewmodel 관련  
 implementation ("androidx.lifecycle:lifecycle-viewmodel-ktx:2.3.1")  
 implementation ("androidx.lifecycle:lifecycle-livedata-ktx:2.3.1")  
 implementation ("androidx.lifecycle:lifecycle-extensions:2.2.0")  
 implementation(libs.androidx.lifecycle.viewmodel.compose)  
  
// Testing  
 testImplementation("junit:junit:4.13.2")  
 androidTestImplementation("androidx.test.ext:junit:1.1.5")  
 androidTestImplementation("androidx.test.espresso:espresso-core:3.5.1")  
  
// Tooling  
 debugImplementation("androidx.compose.ui:ui-tooling:1.5.1")  
 androidTestImplementation("androidx.compose.ui:ui-test-junit4:1.5.1")  
 debugImplementation("androidx.compose.ui:ui-test-manifest:1.5.1")  
**}**

// Top-level build file where you can add configuration options common to all sub-projects/modules.  
plugins **{** alias(libs.plugins.androidApplication) apply false  
 alias(libs.plugins.jetbrainsKotlinAndroid) apply false  
  
**}**buildscript **{** val kotlinVersion = "1.9.0" // 변수 이름을 kotlinVersion으로 통일  
  
  
 repositories **{** google()  
 mavenCentral()  
 **}** dependencies **{** classpath("org.jetbrains.kotlin:kotlin-gradle-plugin:1.9.0") // Kotlin Gradle 플러그인  
 // 추가적인 종속성 필요 시 여기에 추가  
 **}  
}**allprojects **{** tasks.withType<org.jetbrains.kotlin.gradle.tasks.KotlinCompile> **{** kotlinOptions **{** jvmTarget = "1.8"  
 **}  
 }  
}**

pluginManagement **{** repositories **{** gradlePluginPortal()  
 google()  
 mavenCentral()  
 **}  
}**dependencyResolutionManagement **{** *repositoriesMode*.set(RepositoriesMode.*FAIL\_ON\_PROJECT\_REPOS*)  
 repositories **{** google()  
 mavenCentral()  
 **}  
}***rootProject*.*name* = "Toolkit\_management\_mk3"  
include(":app")