package com.example.toolkit\_management\_mk3  
  
import android.annotation.SuppressLint  
import android.content.Context  
import android.os.Bundle  
import android.util.Log  
import android.view.MotionEvent  
import android.view.inputmethod.InputMethodManager  
import android.widget.Button  
import android.widget.EditText  
import android.widget.LinearLayout  
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 addTextMainItem: EditText  
 private lateinit var addTextSubItem: EditText  
 private lateinit var buttonAddItem: Button  
 private var lastItemId = 0  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*item\_adding\_layout*)  
  
 listView = findViewById(R.id.*listViewItems*)  
 addTextMainItem = findViewById(R.id.*AddMainText*)  
 addTextSubItem = findViewById(R.id.*AddSubItem*)  
 buttonAddItem = findViewById(R.id.*buttonAddItem*)  
 adapter = CustomItemAdapter\_1(  
 this,  
 R.layout.*item\_edittext\_layout*,  
 itemList  
 ) // 확인: item\_edittext\_layout 사용  
 listView.*adapter* = adapter  
 listView.*isFocusable* = true  
 listView.*isClickable* = true  
 listView.*isFocusableInTouchMode* = true  
  
 findViewById<LinearLayout>(R.id.*main\_layout*).*apply* **{** *isFocusable* = true  
 *isClickable* = true  
 *isFocusableInTouchMode* = true  
 **}** setupTouchListener()  
  
 loadData()  
  
 buttonAddItem.setOnClickListener **{** manageItemAddition()  
 **}** }  
 @SuppressLint("ClickableViewAccessibility")  
 private fun setupTouchListener() {  
 val mainLayout = findViewById<LinearLayout>(R.id.*main\_layout*)  
  
 mainLayout.setOnTouchListener **{** \_, event **->** if (event.*action* == MotionEvent.*ACTION\_DOWN*) {  
 clearFocusAndHideKeyboard()  
 false  
 } else {  
 false  
 }  
 **}** listView.setOnTouchListener **{** \_, event **->** if (event.*action* == MotionEvent.*ACTION\_DOWN*) {  
 clearFocusAndHideKeyboard()  
 true  
 } else {  
 false  
 }  
 **}** }  
  
 private fun clearFocusAndHideKeyboard() {  
 val focusView = *currentFocus* focusView?.*let* **{  
 it**.clearFocus() // 포커스 제거  
 val imm = getSystemService(Context.*INPUT\_METHOD\_SERVICE*) as InputMethodManager  
 imm.hideSoftInputFromWindow(**it**.*windowToken*, 0) // 키보드 숨김  
 **}** }  
  
 override fun onTouchEvent(event: MotionEvent): Boolean {  
 when (event.*action*) {  
 MotionEvent.*ACTION\_DOWN* -> {  
 // 현재 포커스를 가진 뷰의 ID를 로그로 출력  
 val focusView = *currentFocus* if (focusView != null) {  
 val resourceName = *resources*.getResourceEntryName(focusView.*id*)  
 Log.d("PocketActivity", "Focused View ID: $resourceName")  
 } else {  
 Log.d("PocketActivity", "No view currently has focus")  
 }  
 }  
 }  
 return super.onTouchEvent(event)  
 }  
 private fun manageItemAddition() {  
 loadData()  
 val mainItemText = addTextMainItem.*text*.toString()  
 val subItemText = addTextSubItem.*text*.toString()  
  
 if (mainItemText.*isNotBlank*()) {  
 val newItem = Item(++lastItemId, mainItemText, subItemText)  
 itemList.add(newItem)  
 adapter.updateItems(itemList)  
 addTextMainItem.*text*.clear()  
 addTextSubItem.*text*.clear()  
 saveItems()  
 } else {  
 Log.d("PocketActivity", "Main text is blank")  
 }  
 }  
  
 private fun loadData() {  
 val sharedPreferences = getSharedPreferences("AppPrefs", Context.*MODE\_PRIVATE*)  
 val json = sharedPreferences.getString("items", null)  
 if (json != null) {  
 itemList = Gson().fromJson(json, object : TypeToken<MutableList<Item>>() {}.*type*)  
 Log.d("PocketActivity", "Loaded items: ${itemList.size}")  
 } else {  
 Log.d("PocketActivity", "No items found in SharedPreferences")  
 itemList.clear()  
 }  
 adapter.updateItems(itemList)  
 }  
  
 private fun saveItems() {  
 val json = Gson().toJson(itemList)  
 val sharedPreferences = getSharedPreferences("AppPrefs", Context.*MODE\_PRIVATE*)  
 *with*(sharedPreferences.edit()) **{** putString("items", json)  
 apply()  
 **}** }  
}

포커스빼고 작동하는 버전.

package com.example.toolkit\_management\_mk3  
  
import android.content.Context  
import android.util.Log  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
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  
import kotlinx.coroutines.CoroutineScope  
import kotlinx.coroutines.Dispatchers  
import kotlinx.coroutines.launch  
  
class CustomItemAdapter\_1(  
 private val context: Context,  
 private val resource: Int,  
 items: MutableList<Item>  
) : ArrayAdapter<Item>(context, resource, items) {  
  
 var items: MutableList<Item> = items  
 set(value) {  
 **field** = value  
 notifyDataSetChanged()  
 }  
  
 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.*editMainText*),  
 view.findViewById(R.id.*editSubItem*),  
 view.findViewById(R.id.*buttonDelete*)  
 )  
 view.*tag* = holder  
 } else {  
 view = convertView  
 holder = view.*tag* as ViewHolder  
 }  
  
 val item = items[position]  
 holder.editTextItem.setText(item.mainText)  
 holder.editTextSubItem.setText(item.subItems)  
 holder.editTextItem.setSelection(item.mainText.length)  
  
 setListeners(holder, position)  
  
 return view  
 }  
  
 fun updateItems(newItems: MutableList<Item>) {  
 Log.d("CustomItemAdapter\_1", "Updating items in adapter: ${newItems.size}")  
 if (newItems.*isNotEmpty*()) {  
 this.items.clear()  
 this.items.addAll(newItems)  
 } else {  
 this.items.clear()  
 }  
 notifyDataSetChanged()  
 Log.d("CustomItemAdapter\_1", "Items updated in adapter")  
 }  
  
 private fun setListeners(holder: ViewHolder, position: Int) {  
 holder.editTextItem.setOnFocusChangeListener **{** v, hasFocus **->** if (!hasFocus) {  
 saveData(position, holder.editTextItem.*text*.toString(), holder.editTextSubItem.*text*.toString())  
 }  
 Log.d("CustomItemAdapter\_1", "Focus changed in Main Text: Position $position, Has Focus: $hasFocus")  
 **}** holder.editTextSubItem.setOnFocusChangeListener **{** v, hasFocus **->** if (!hasFocus) {  
 // 포커스를 잃었을 때 로직 실행  
 Log.d("CustomItemAdapter", "SubText lost focus at position $position")  
 saveData(position, holder.editTextItem.*text*.toString(), holder.editTextSubItem.*text*.toString())  
  
 // 다음과 같이 mainText로 포커스를 강제로 이동하는 로직이 있다면 제거  
 // holder.editTextItem.requestFocus()  
 } else {  
 Log.d("CustomItemAdapter", "SubText got focus at position $position")  
 }  
 **}** holder.deleteButton.setOnClickListener **{** items.removeAt(position)  
 notifyDataSetChanged()  
 saveItemsAsync()  
 Log.d("CustomItemAdapter\_1", "Item deleted at position: $position")  
 **}** }  
  
 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()  
 saveItemsAsync()  
 }  
 }  
  
 private fun saveItemsAsync() {  
 *CoroutineScope*(Dispatchers.IO).*launch* **{** val json = Gson().toJson(items)  
 sharedPreferences.edit().putString("items", json).apply()  
 Log.d("CustomItemAdapter\_1", "Data saved asynchronously: $json")  
 **}** }  
  
 private class ViewHolder(val editTextItem: EditText, val editTextSubItem: EditText, val deleteButton: Button)  
}

package com.example.toolkit\_management\_mk3  
  
import android.content.Context  
import android.graphics.Color  
import android.util.Log  
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.com.example.toolkit\_management\_mk3.ColorViewModel  
import com.example.toolkit\_management\_mk3.models.Item  
  
class CustomItemAdapter\_2(  
 private val context: Context,  
 private val resource: Int,  
 private var items: MutableList<Item>,  
 private val viewModel: ColorViewModel // ViewModel 인스턴스가 전달되어야 함  
) : ArrayAdapter<Item>(context, resource, items) {  
  
 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)  
 if (textViewItem == null || subTextView == null) {  
 Log.e("CustomItemAdapter\_2", "TextViews are null at position $position")  
 }  
  
 if (item != null) {  
 textViewItem?.text = item.mainText  
 subTextView?.text = item.subItems  
  
 val color = getColorForState(viewModel.colorState.value, item.id)  
 textViewItem?.setTextColor(color)  
 subTextView?.setTextColor(color)  
  
 textViewItem?.setOnClickListener **{** viewModel.updateColorState(context, item.id) // ViewModel의 메서드를 호출  
 **}** } else {  
 Log.e("CustomItemAdapter\_2", "Item is null at position $position")  
 textViewItem?.text = "Invalid item"  
 subTextView?.text = "Invalid sub item"  
 }  
  
 return itemView  
 }  
  
 private fun getColorForState(colorState: Map<Int, Int>?, itemId: Int): Int {  
 return when (colorState?.getOrDefault(itemId, 0)) {  
 1 -> Color.BLUE  
 2 -> Color.RED  
 else -> Color.BLACK  
 }  
 }  
  
 fun updateColorState(newColorState: Map<Int, Int>) {  
 notifyDataSetChanged() // UI 갱신  
 }  
  
 fun updateItems(newItems: MutableList<Item>) {  
 Log.d("CustomItemAdapter\_2", "Before update, current items: ${items.size}, new items: ${newItems.size}")  
 if (newItems.isNotEmpty()) {  
 this.items.clear()  
 this.items.addAll(newItems)  
 } else {  
 Log.d("CustomItemAdapter\_2", "Received empty items list for update.")  
 }  
 notifyDataSetChanged()  
 Log.d("CustomItemAdapter\_2", "After update, current items: ${items.size}")  
 }  
}

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 lateinit var adapter: CustomItemAdapter\_2  
 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)  
  
 adapter = CustomItemAdapter\_2(this, R.layout.display\_item\_layout, itemList, viewModel)  
 listView.adapter = adapter  
  
 loadData()  
  
 viewModel.colorState.observe(this, Observer **{** updatedColorState **->** adapter.updateColorState(updatedColorState)  
 adapter.notifyDataSetChanged()  
 **}**)  
  
 val buttonEdit: Button = findViewById(R.id.buttonEdit)  
 buttonEdit.setOnClickListener **{** showDisplayDialog()  
 **}** }  
  
 override fun onResume() {  
 super.onResume()  
 loadData() // 데이터를 다시 로드하여 어댑터에 최신 정보를 제공  
 }  
  
  
 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<MutableList<Item>>() {}.type)  
 adapter.updateItems(itemList) // 데이터 로드 후 어댑터에 아이템 업데이트  
 Log.d("DisplayActivity", "Items updated in adapter")  
 } else {  
 Log.d("DisplayActivity", "No items found in SharedPreferences, initializing empty list.")  
 itemList.clear()  
 adapter.updateItems(itemList) // 데이터가 없을 경우, 어댑터에 빈 리스트 설정  
 }  
 adapter.notifyDataSetChanged()  
 }  
  
 fun showDisplayDialog() {  
 val fragment = DisplayDialogFragment.newInstance()  
 fragment.show(supportFragmentManager, "displayDialog")  
 }  
}

package com.example.toolkit\_management\_mk3  
  
import android.content.Intent  
import android.graphics.Color  
import android.graphics.drawable.ColorDrawable  
import android.os.Bundle  
import android.util.Log  
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.models.Item  
  
class DisplayDialogFragment : DialogFragment() {  
 private lateinit var listView: ListView  
 private var adapter: CustomItemAdapter\_2? = null  
 private var itemList: MutableList<Item> = mutableListOf()  
 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 = ViewModelProvider(requireActivity()).get(ColorViewModel::class.java)  
 viewModel.loadColorState(requireContext())  
 viewModel.loadItems(requireContext())  
 }  
  
 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)  
 adapter = CustomItemAdapter\_2(requireContext(), R.layout.custom\_dispactview, itemList, viewModel)  
 listView.adapter = adapter  
  
 observeViewModel()  
  
 setupItemListeners()  
  
 val closeButton: Button = view.findViewById(R.id.buttonClose)  
 closeButton.setOnClickListener **{** dismiss()  
 startActivity(Intent(context, DisplayActivity::class.java))  
 **}** val editButton: Button = view.findViewById(R.id.buttonEdit)  
 editButton.setOnClickListener **{** startActivity(Intent(activity, PocketActivity::class.java)) **}** }  
  
 private fun observeViewModel() {  
 viewModel.items.observe(viewLifecycleOwner, **{** items **->** itemList.clear()  
 itemList.addAll(items)  
 adapter?.notifyDataSetChanged()  
 **}**)  
  
 viewModel.colorState.observe(viewLifecycleOwner) **{** updatedColorState **->** itemList.forEach **{** item **->** item.color = updatedColorState[item.id] ?: Color.BLACK  
 **}** adapter?.notifyDataSetChanged() // 색상이 업데이트된 후 화면에 반영  
 **}** }  
  
 private fun setupItemListeners() {  
 listView.setOnItemClickListener **{** \_, \_, position, \_ **->** val item = itemList.getOrNull(position)  
 item?.let **{** Log.d("DisplayDialogFragment", "Item clicked: ${it.mainText}")  
 viewModel.updateColorState(requireContext(), it.id)  
 adapter?.notifyDataSetChanged() // Ensure the adapter refreshes the view  
 **}  
 }** }  
  
 override fun onStart() {  
 super.onStart()  
 dialog?.window?.apply **{** setLayout(ViewGroup.LayoutParams.MATCH\_PARENT, ViewGroup.LayoutParams.WRAP\_CONTENT)  
 setBackgroundDrawable(ColorDrawable(Color.TRANSPARENT))  
 **}** loadData()  
 }  
  
 private fun loadData() {  
 // 아이템과 색상 상태를 뷰모델에서 로드합니다.  
 viewModel.loadColorState(requireContext())  
 viewModel.loadItems(requireContext())  
  
 // 뷰모델 관찰자를 설정합니다.  
 observeViewModel()  
 }  
}

package com.example.toolkit\_management\_mk3  
  
import android.os.Bundle  
import android.widget.ArrayAdapter  
import android.widget.ListView  
import androidx.activity.viewModels  
import androidx.appcompat.app.AppCompatActivity  
import androidx.lifecycle.Observer  
import com.example.toolkit\_management\_mk3.com.example.toolkit\_management\_mk3.ColorViewModel  
  
class CheckActivity : AppCompatActivity() {  
 private val viewModel: ColorViewModel by viewModels()  
  
  
 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.  
 viewModel.loadColorState(this)  
 viewModel.loadItems(this)  
  
 // 로그 확인을 위해 호출 확인 로그 추가  
 println("Color State and Items are being loaded in CheckActivity")  
  
  
 // 모든 아이템을 로드하고 빨간색인 아이템만 필터링합니다.  
 viewModel.items.observe(this, Observer **{** items **->** viewModel.colorState.value?.let **{** colorState **->** val redItems = items.filter **{** item **->** colorState.getOrDefault(item.id, 0) == 2 **}** val adapter = ArrayAdapter(this, android.R.layout.simple\_list\_item\_1, redItems.map **{** it.mainText **}**)  
 listViewRedItems.adapter = adapter  
 **}  
 }**)  
 }  
}

import android.graphics.Color  
import android.graphics.drawable.ColorDrawable  
import android.os.Bundle  
import android.util.Log  
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 androidx.lifecycle.ViewModelProvider  
import com.example.toolkit\_management\_mk3.R  
import com.example.toolkit\_management\_mk3.com.example.toolkit\_management\_mk3.ColorViewModel  
  
class CheckDialogFragment : DialogFragment() {  
 private lateinit var listView: ListView  
 private lateinit var adapter: ArrayAdapter<String>  
 private lateinit var viewModel: ColorViewModel  
  
 companion object {  
 fun newInstance() = CheckDialogFragment()  
 }  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setStyle(DialogFragment.STYLE\_NORMAL, R.style.CustomDialogStyle)  
 }  
  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?  
 ): View? {  
 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)  
 adapter = ArrayAdapter(requireContext(), R.layout.custom\_textview, R.id.customTextView, mutableListOf<String>())  
 listView.adapter = adapter  
  
 viewModel = ViewModelProvider(requireActivity()).get(ColorViewModel::class.java)  
 observeViewModel()  
 viewModel.loadColorState(requireContext())  
 viewModel.loadItems(requireContext())  
 // 로그 확인을 위해 호출 확인 로그 추가  
 println("Color State and Items are being loaded in CheckDialogFragment")  
 }  
  
 private fun observeViewModel() {  
 viewModel.items.observe(viewLifecycleOwner) { items ->  
 viewModel.colorState.value?.let { colorState ->  
 val redItems = items.filter { item -> colorState.get(item.id) == 2 }  
 Log.d("CheckDialogFragment", "Filtered Red Items: ${redItems.map { it.mainText }}")  
 adapter.clear()  
 adapter.addAll(redItems.map { it.mainText })  
 adapter.notifyDataSetChanged()  
 }  
 }  
 }  
  
 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 androidx.lifecycle.MutableLiveData  
import androidx.lifecycle.ViewModel  
import com.example.toolkit\_management\_mk3.models.Item  
import com.google.gson.Gson  
import com.google.gson.reflect.TypeToken  
  
class ColorViewModel : ViewModel() {  
 val colorState = MutableLiveData<MutableMap<Int, Int>>() // 상태 변경을 위한 LiveData  
 val items = MutableLiveData<List<Item>>() // 아이템 리스트를 관리하는 LiveData  
  
 // 색상 상태를 로드하는 함수  
 fun loadColorState(context: Context) {  
 val prefs = context.getSharedPreferences("AppPrefs", Context.*MODE\_PRIVATE*)  
 val colorStateJson = prefs.getString("colorState", "{}")  
 *println*("Color State JSON: $colorStateJson") // 로그 추가  
 if (colorStateJson.*isNullOrEmpty*()) {  
 *println*("Color State JSON is null or empty")  
 } else {  
 val type = object : TypeToken<MutableMap<Int, Int>>() {}.*type* colorState.*value* = Gson().fromJson(colorStateJson, type)  
 *println*("Loaded color state: ${colorState.*value*}") // 로그 확인  
 }  
 }  
  
 // 색상 상태를 업데이트하는 함수  
 fun updateColorState(context: Context, itemId: Int) {  
 val currentState = colorState.*value* ?: *mutableMapOf*()  
 val newState = (currentState.getOrDefault(itemId, 0) + 1) % 3  
 currentState[itemId] = newState  
 colorState.*value* = currentState // LiveData를 통해 UI 업데이트  
 saveColorState(context, currentState)  
 }  
  
 // 변경된 색상 상태를 저장하는 함수  
 private fun saveColorState(context: Context, colorState: MutableMap<Int, Int>) {  
 val prefs = context.getSharedPreferences("AppPrefs", Context.*MODE\_PRIVATE*)  
 *with*(prefs.edit()) **{** putString("colorState", Gson().toJson(colorState))  
 apply()  
 **}** }  
  
 // 아이템을 로드하는 함수  
 fun loadItems(context: Context) {  
 val sharedPreferences = context.getSharedPreferences("AppPrefs", Context.*MODE\_PRIVATE*)  
 val itemsJson = sharedPreferences.getString("items", null)  
 *println*("Items JSON: $itemsJson") // 로그 추가  
 if (itemsJson.*isNullOrEmpty*()) {  
 *println*("Items JSON is null or empty")  
 } else {  
 val itemType = object : TypeToken<List<Item>>() {}.*type* items.*value* = Gson().fromJson(itemsJson, itemType)  
 *println*("Loaded items: ${items.*value*?.size}") // 로그 확인  
 }  
 }  
}

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