

# برنامه نویسی دستگاه های سیار (CE364)

## جلسه هفدهم: ذخیره اطلاعات در پایگاه داده

**سجاد شیرعلی شمرضا**

**پاییز 1401**

**شنبه، 3 دی 1401**

● بخش مرتبط با این جلسه:

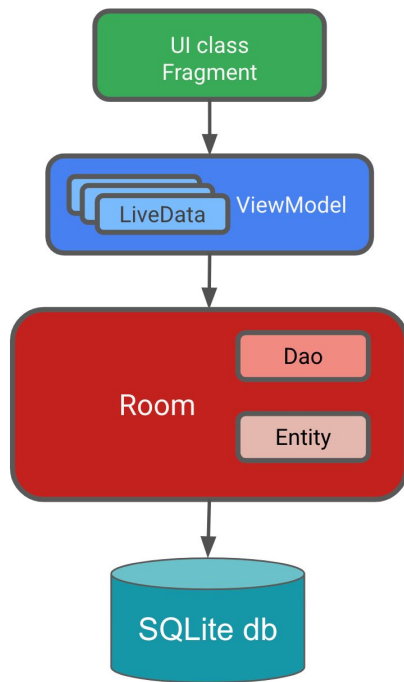
- Unit 5: Data persistence:
  - Pathway 2: Use Room for data persistence



سوال؟

# تعریف دسترسی به پایگاه داده

# ذخیره حالت



- نیاز به ذخیره اطلاعات در بسیاری از برنامه ها
  - ذخیره لیست کارهایی که کاربر میخواهد انجام دهد
  - ذخیره شماره آخرین صفحه خوانده شده
- کتابخانه اتاق (Room)
  - یک لایه انتزاعی بر روی پایگاه داده SQLite
    - ساده سازی ایجاد، تنظیم، و استفاده از پایگاه داده
    - بررسی عبارت های SQL در زمان کامپایل

# برنامه نهایی: موجودی انبار

Inventory		
ITEM	PRICE	QUANTITY IN STOCK
Apples	1.5	83
Bananas	2.5	0
Blueberry	5.0	89
Dragon Fruit	3.0	23
Grapes	4.0	40
Oranges	6.0	30
Strawberry	14.0	90
Water Melon	4.0	23

Add Product

SAVE

CANCEL

Edit Product

Grapes

4.0

40

SAVE

CANCEL

123-  
456-  
789-  
,0.✓

# ساختار برنامه اولیه

The image shows two mobile application screens. The left screen, titled 'Inventory', features a purple header bar. Below it is a table with three columns: 'ITEM', 'PRICE', and 'QUANTITY IN STOCK'. The table body is currently empty. A teal circular button with a white plus sign is positioned at the bottom right of the screen. The right screen, titled 'Add Item', also has a purple header bar. It contains three input fields: 'Item Name \*', 'Item Price \*', and 'Quantity in Stock \*'. Each field has a red asterisk indicating a required field. Below these fields is a purple button labeled 'SAVE'.

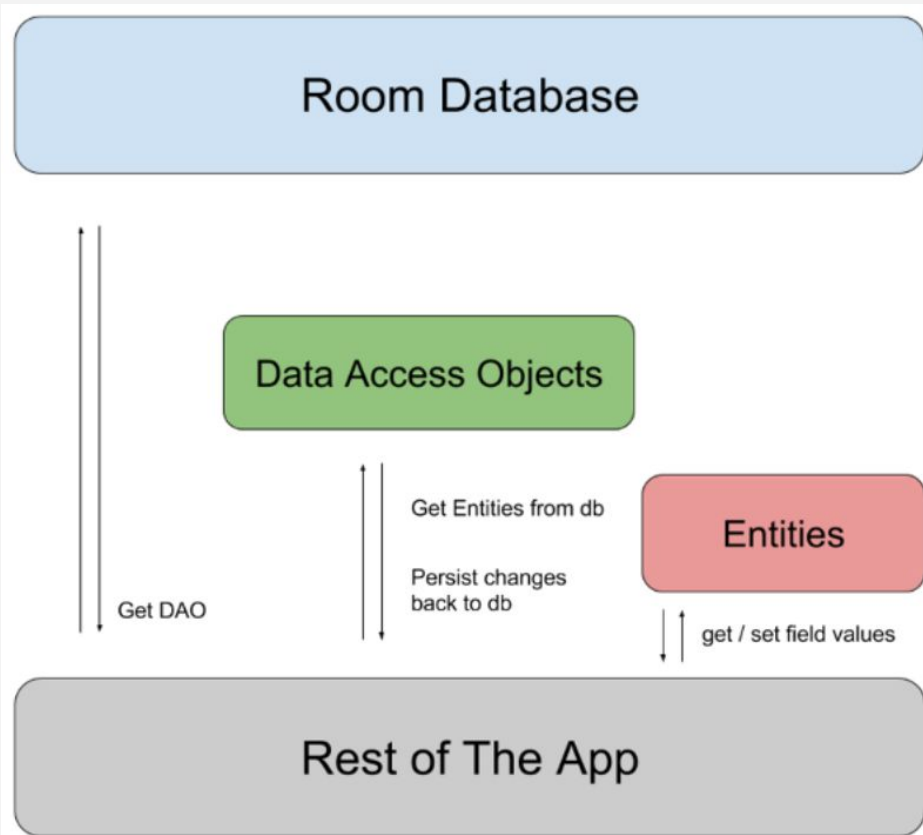
- برنامه هیچ اطلاعاتی از موجودی نشان نمی دهد
- دکمه شناور انجام کار (FAB: Floating Action Button) در سمت راست پایین
  - بازکردن یک صفحه جدید برای وارد کردن اطلاعات یک جنس جدید
- ذخیره کردن کار نمی کند

## بخش های اصلی برنامه اولیه

- فعالیت اصلی: `main_activity.xml`
  - میزبانی از قطعات مختلف برنامه
  - ایجاد اشیای مختلف مورد نیاز برای جابجایی بین قطعات
- قطعه نمایش موجودی انبار: `item_list_fragment.xml` و `ItemListFragment.kt`
  - یک مدیر نمای `RecyclerView` و دکمه شناور انجام کار
- قطعه اضافه کردن یک شی جدید: `fragment_add_item.xml` و `AddItemFragment.kt`
  - مخفی کردن صفحه کلید هنگام اتمام قطعه



# اجرای مختلف مورد نیاز برای استفاده از Room



- شی پایگاه داده: نقطه اصلی اتصال برنامه به پایگاه داده
- کلاس دسترسی به داده (DAO)
  - ارائه توابع استفاده از پایگاه داده
- موجودیت (Data Entity)
  - نماینده داده برای استفاده در برنامه

# رابطه جدول پایگاه داده و کلاس موجودیت داده

**Entity fields**

id	Name	Price	Quantity
1	Apples	4.50	200
2	Bananas	1.99	440
3	Strawberry	7.00	580
4	Oranges	6.00	30
...	...	...	...
...	...	...	...
...	...	...	...

Table name: **Item** → Entity class name

## تعریف موجودید (Entity)

- استفاده از حاشیه نویسی برای مشخص کردن این که کلاس داده، موجودیت است:
- @Entity
  - در نظر گرفته شدن یک جدول برای هر موجودیت
  - در نظر گرفته شدن یک ستون به ازای ویژگی (field)
  - نیاز به داشتن یک کلید اصلی
    - عدم امکان تغییر آن پس از ثبت در پایگاه داده
    - بایستی یکتا باشد

## کلاس داده (Data Class)

- برای نگه داری داده در زبان کاتلین
- مشخص شدن با کلمه کلیدی data قبل از کلمه class
- ایجاد خودکار توابعی همچون copy، toString و equals توسط کامپایلر
- شرط های لازم برای کلاس داده
  - حداقل یک پارامتر برای سازنده اصلی
  - تعیین اینکه هریک از پارامترهای ورودی val هستند یا var
  - کلاس نمی تواند abstract, open, sealed و یا inner باشد.

```
// Example data class with 2 properties.
```

```
data class User(val first_name: String, val last_name: String){  
}
```

# تعریف کلاس عنصر

```
import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity
data class Item(
    @PrimaryKey(autoGenerate = true)
    val id: Int = 0,
    @ColumnInfo(name = "name")
    val itemName: String,
    @ColumnInfo(name = "price")
    val itemPrice: Double,
    @ColumnInfo(name = "quantity")
    val quantityInStock: Int
)
```

- ایجاد یک بسته جدید به نام data
- اضافه کردن کلاس عنصر در این بسته
- تعریف ویژگی ها

# تعریف کلاس دسترسی به داده

- هدف: جدا کردن بخش دسترسی به پایگاه داده از سایر برنامه
  - در راستای هدف "یک مسئولیت"
- مخفی کردن پیچیدگی مربوط به دسترسی به پایگاه داده



# حاشیه نویسی برای کلاس دسترسی به داده

- ارائه حاشیه نویسی برای تولید خودکار کد

- @Insert: درج یک عنصر جدید در جدول

```
@Insert(onConflict = OnConflictStrategy.IGNORE)
suspend fun insert(item: Item)
```

- @Delete: حذف یک عنصر

```
@Delete
suspend fun delete(item: Item)
```

- @Update: به روز کردن یک عنصر

```
@Update
suspend fun update(item: Item)
```

- @Query: عبارت درخواست برای اعمال دیگر

```
@Query("SELECT * from item WHERE id = :id")
fun getItem(id: Int): Flow<Item>
```

# تعریف کلاس پایگاه داده

```
import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase

@Database(entities = [Item::class], version = 1, exportSchema = false)
abstract class ItemRoomDatabase : RoomDatabase() {

    abstract fun itemDao(): ItemDao

    companion object {
        @Volatile
        private var INSTANCE: ItemRoomDatabase? = null
        fun getDatabase(context: Context): ItemRoomDatabase {
            return INSTANCE ?: synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    ItemRoomDatabase::class.java,
                    "item_database"
                )
                    .fallbackToDestructiveMigration()
                    .build()
                INSTANCE = instance
                return instance
            }
        }
    }
}
```



# استفاده از کلاس پایگاه داده

- تغییر برنامه اصلی و تعریف یک نمونه از پایگاه داده

```
import android.app.Application
import com.example.inventory.data.ItemRoomDatabase

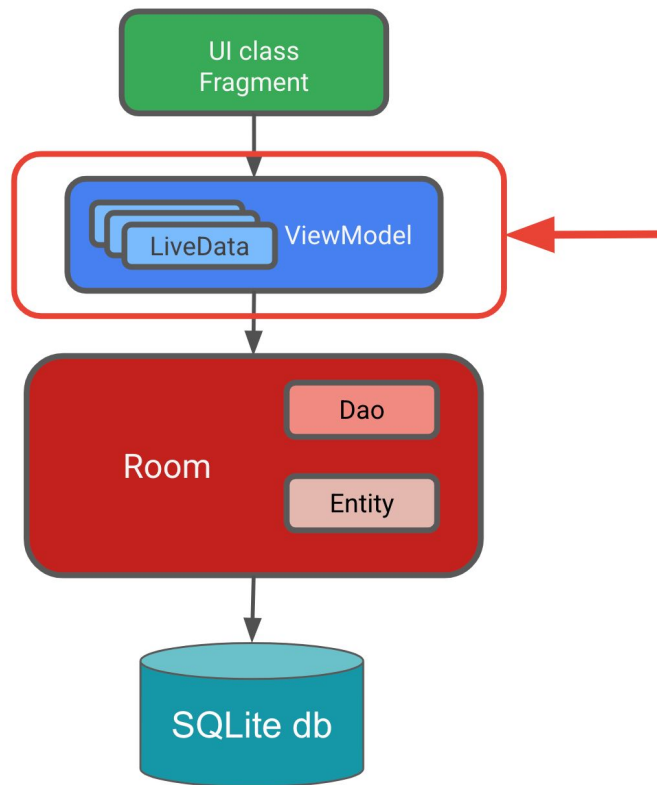
class InventoryApplication : Application(){
    val database: ItemRoomDatabase by lazy { ItemRoomDatabase.getDatabase(this) }
}
```



سوال؟

استفاده از اطلاعات و نمایش آنها

# جایگاه مدل نما



# تعریف مدل نما

- تعریف کلاس جدید

```
class InventoryViewModel(private val itemDao: ItemDao) : ViewModel() {}
```

- تعریف کلاس کارخانه

```
class InventoryViewModelFactory(private val itemDao: ItemDao) : ViewModelProvider.Factory {  
}
```

- تعریف تابع گرفتن کتابخانه

```
override fun <T : ViewModel?> create(modelClass: Class<T>): T {  
    if (modelClass.isAssignableFrom(InventoryViewModel::class.java)) {  
        @Suppress("UNCHECKED_CAST")  
        return InventoryViewModel(itemDao) as T  
    }  
    throw IllegalArgumentException("Unknown ViewModel class")  
}
```

# پیاده سازی توابع مدل نما

```
private fun insertItem(item: Item) {  
    viewModelScope.launch {  
        itemDao.insert(item)  
    }  
}
```

- تابع درج عنصر

- ایجاد یک عنصر

```
private fun getItemEntry(itemName: String, itemPrice: String, itemCount: String): Item {  
    return Item(  
        itemName = itemName,  
        itemPrice = itemPrice.toDouble(),  
        quantityInStock = itemCount.toInt()  
    )  
}  
  
fun addItem(itemName: String, itemPrice: String, itemCount: String) {  
    val newItem = getItemEntry(itemName, itemPrice, itemCount)  
    insertItem(newItem)  
}
```

# استفاده از مدل نما در برنامه

- تعریف یک نمونه از مدل نما در فایل AddItemFragment.kt

```
private val viewModel: InventoryViewModel by activityViewModels {  
    InventoryViewModelFactory(  
        (activity?.application as InventoryApplication).database  
            .itemDao()  
    )  
}
```

# توابع اضافه کردن یک عنصر جدید

```
fun isEntryValid(itemName: String, itemPrice: String, itemCount: String): Boolean {  
    if (itemName.isBlank() || itemPrice.isBlank() || itemCount.isBlank()) {  
        return false  
    }  
    return true  
}  
  
private fun isEntryValid(): Boolean {  
    return viewModel.isEntryValid(  
        binding.itemName.text.toString(),  
        binding.itemPrice.text.toString(),  
        binding.itemCount.text.toString()  
    )  
}  
  
private fun addNewItem() {  
    if (isEntryValid()) {  
        viewModel.addNewItem(  
            binding.itemName.text.toString(),  
            binding.itemPrice.text.toString(),  
            binding.itemCount.text.toString(),  
        )  
        val action = AddItemFragmentDirections.actionAddItemFragmentToItemListFragment  
        findNavController().navigate(action)  
    }  
}
```



## ذخیره هنگام فشردن کلید ذخیره

```
override fun onCreateView(view: View, savedInstanceState: Bundle?) {  
    super.onCreateView(view, savedInstanceState)  
    binding.saveAction.setOnClickListener {  
        addNewItem()  
    }  
}
```



سوال؟

# نمایش عناصر موجود

# برنامه نهایی

## Inventory

ITEM	PRICE	QUANTITY IN STOCK
Apples	\$43.00	54
Bananas	\$543.00	23
Blueberry	\$43.00	0
Honey	\$4.00	23
Oranges	\$45.00	123
Raspberry	\$6.00	94
Strawberry	\$5.00	5
Test	\$54.00	34
Tomatoes	\$5.00	32



# تعريف تابع توسعه

```
class Square(val side: Double){
    fun area(): Double{
        return side * side;
    }
}

// Extension function to calculate the perimeter of the square
fun Square.perimeter(): Double{
    return 4 * side;
}

// Usage
fun main(args: Array<String>){
    val square = Square(5.5);
    val perimeterValue = square.perimeter()
    println("Perimeter: $perimeterValue")
    val areaValue = square.area()
    println("Area: $areaValue")
}
```

## نمایش قیمت به صورت درست

- توسعه کلاس عنصر
- عدم تغییر نوع ذخیره شده در آن

```
fun Item.getFormattedPrice(): String =  
    NumberFormat.getCurrencyInstance().format(itemPrice)
```

# تعریف مبدل برای نمایش (بخش 1)

```
package com.example.inventory

import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.recyclerview.widget.DiffUtil
import androidx.recyclerview.widget.ListAdapter
import androidx.recyclerview.widget.RecyclerView
import com.example.inventory.data.Item
import com.example.inventory.data.getFormattedPrice
import com.example.inventory.databinding.ItemListItemBinding

/**
 * [ListAdapter] implementation for the recyclerview.
 */

class ItemListAdapter(private val onItemClick: (Item) -> Unit) :
    ListAdapter<Item, ItemListAdapter.ItemViewHolder>(DiffCallback) {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ItemViewHolder {
        return ItemViewHolder(
            ItemListItemBinding.inflate(
                LayoutInflater.from(
                    parent.context
                )
            )
        )
    }
}
```

## تعریف مبدل برای نمایش (بخش 2)

```
override fun onBindViewHolder(holder: ItemViewHolder, position: Int) {
    val current = getItem(position)
    holder.itemView.setOnClickListener {
        onItemClick(current)
    }
    holder.bind(current)
}

class ItemViewHolder(private var binding: ItemListItemBinding) :
    RecyclerView.ViewHolder(binding.root) {

    fun bind(item: Item) {

    }
}

companion object {
    private val DiffCallback = object : DiffUtil.ItemCallback<Item>() {
        override fun areItemsTheSame(oldItem: Item, newItem: Item): Boolean {
            return oldItem === newItem
        }

        override fun areContentsTheSame(oldItem: Item, newItem: Item): Boolean {
            return oldItem.itemName == newItem.itemName
        }
    }
}
```



## تابع متصل کردن مقادیر

```
fun bind(item: Item) {  
    binding.apply {  
        itemName.text = item.itemName  
        itemPrice.text = item.getFormattedPrice()  
        itemQuantity.text = item.quantityInStock.toString()  
    }  
}
```

## استفاده از مبدل لیست،

```
val allItems: LiveData<List<Item>> = itemDao.getItems().asLiveData()
```

```
override fun onCreateView(view: View, savedInstanceState: Bundle?) {  
    super.onCreateView(view, savedInstanceState)
```

```
    val adapter = ItemListAdapter {  
        }  
    binding.recyclerView.adapter = adapter
```

```
    viewModel.allItems.observe(this.viewLifecycleOwner) { items ->
```

```
        items.let {  
            adapter.submitList(it)  
        }  
    }
```

```
    binding.recyclerView.layoutManager = LinearLayoutManager(this.context)
```

```
    binding.floatingActionButton.setOnClickListener {
```

```
        val action = ItemListFragmentDirections.actionItemListFragmentToAddItemFragment(  
            getString(R.string.add_fragment_title)  
        )
```

```
        this.findNavController().navigate(action)
```

```
    }
```

```
}
```

# تعریف مبدل جهت استفاده

- تعریف مبدل

```
val adapter = ItemListAdapter {  
    val action = ItemListFragmentDirections.actionItemListFragmentToItemDetailFragment(  
        this.findNavController().navigate(action)  
    )  
}
```

## نمایش عنصر

```
fun retrieveItem(id: Int): LiveData<Item> {  
    return itemDao.getItem(id).asLiveData()  
}
```

```
private fun bind(item: Item) {  
    binding.itemName.text = item.itemName  
    binding.itemPrice.text = item.getFormattedPrice()  
    binding.itemCount.text = item.quantityInStock.toString()  
}
```

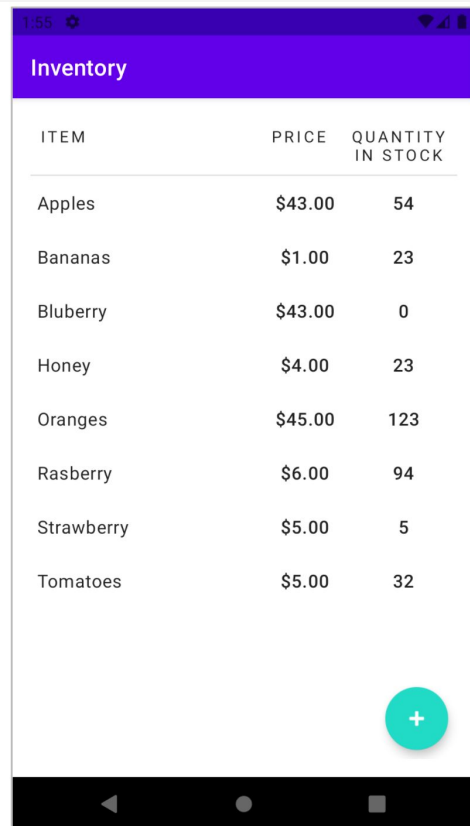
```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
    super.onViewCreated(view, savedInstanceState)  
    val id = navigationArgs.itemId  
    viewModel.retrieveItem(id).observe(this.viewLifecycleOwner) { selectedItem ->  
        item = selectedItem  
        bind(item)  
    }  
}
```



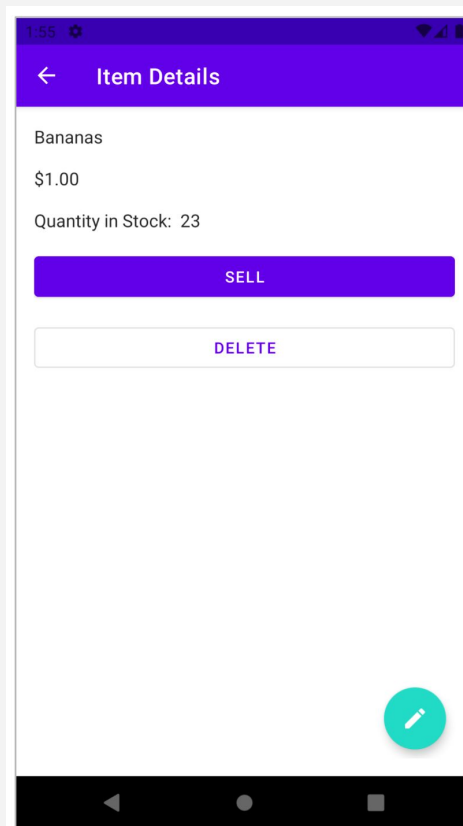
سوال؟

# فروش تعدادی از عناصر موجود

# برنامه نهایی

A screenshot of a mobile application titled 'Inventory'. It features a purple header bar with the title. Below the header is a table with three columns: 'ITEM', 'PRICE', and 'QUANTITY IN STOCK'. The table lists various fruits and their prices and stock levels. At the bottom right, there is a teal circular button with a white plus sign. The status bar at the top shows the time as 1:55 and various icons.

ITEM	PRICE	QUANTITY IN STOCK
Apples	\$43.00	54
Bananas	\$1.00	23
Bluberry	\$43.00	0
Honey	\$4.00	23
Oranges	\$45.00	123
Raspberry	\$6.00	94
Strawberry	\$5.00	5
Tomatoes	\$5.00	32

A screenshot of a mobile application titled 'Item Details'. It features a purple header bar with a back arrow and the title. The main content area displays details for 'Bananas', including the price '\$1.00' and 'Quantity in Stock: 23'. There are two buttons: a solid purple 'SELL' button and a white 'DELETE' button with a purple border. At the bottom right, there is a teal circular button with a white pencil icon. The status bar at the top shows the time as 1:55 and various icons.

← Item Details

Bananas

\$1.00

Quantity in Stock: 23

SELL

DELETE

## پیاده سازی فروش

```
private fun updateItem(item: Item) {
    viewModelScope.launch {
        itemDao.update(item)
    }
}

fun sellItem(item: Item) {
    if (item.quantityInStock > 0) {
        // Decrease the quantity by 1
        val newItem = item.copy(quantityInStock = item.quantityInStock - 1)
        updateItem(newItem)
    }
}

fun isStockAvailable(item: Item): Boolean {
    return (item.quantityInStock > 0)
}

private fun bind(item: Item) {
    binding.apply {
        ...
        sellItem.isEnabled = viewModel.isStockAvailable(item)
        sellItem.setOnClickListener { viewModel.sellItem(item) }
    }
}
```



## حذف عنصر

```
fun deleteItem(item: Item) {  
    viewModelScope.launch {  
        itemDao.delete(item)  
    }  
}  
  
private fun deleteItem() {  
    viewModel.deleteItem(item)  
    findNavController().navigateUp()  
}  
  
private fun showConfirmationDialog() {  
    MaterialAlertDialogBuilder(requireContext())  
        ...  
        .setPositiveButton(getString(R.string.yes)) { _, _ ->  
            deleteItem()  
        }  
        .show()  
}
```

```
private fun bind(item: Item) {  
    binding.apply {  
        ...  
        deleteItem.setOnClickListener { showConfirmationDialog() }  
    }  
}
```

### Attention

Are you sure you want to delete?

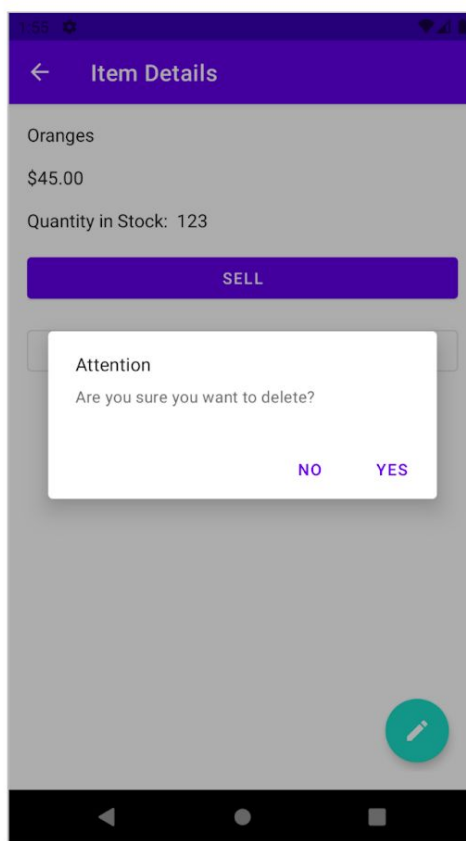
NO

YES

# نسخه نهایی حذف

A screenshot of an Android application titled 'Inventory'. It features a purple header bar with the title. Below the header is a table with three columns: 'ITEM', 'PRICE', and 'QUANTITY IN STOCK'. The table lists various fruits and their prices and stock quantities. At the bottom right, there is a green circular button with a white plus sign. The status bar at the top shows the time as 1:55 and various icons.

ITEM	PRICE	QUANTITY IN STOCK
Apples	\$43.00	54
Bananas	\$1.00	21
Bluberry	\$43.00	0
Honey	\$4.00	23
Oranges	\$45.00	123
Raspberry	\$6.00	92
Strawberry	\$5.00	5
Tomatoes	\$5.00	32

A screenshot of an Android application titled 'Item Details'. It has a purple header bar with a back arrow and the title. The main content area shows details for 'Oranges', including a price of '\$45.00' and 'Quantity in Stock: 123'. Below this is a purple 'SELL' button. A white dialog box is centered on the screen with the title 'Attention' and the text 'Are you sure you want to delete?'. The dialog has two buttons: 'NO' and 'YES'. At the bottom right, there is a green circular button with a white plus sign. The status bar at the top shows the time as 1:55 and various icons.

← Item Details

Oranges

\$45.00

Quantity in Stock: 123

SELL

Attention

Are you sure you want to delete?

NO YES

A screenshot of an Android application titled 'Inventory'. It features a purple header bar with the title. Below the header is a table with three columns: 'ITEM', 'PRICE', and 'QUANTITY IN STOCK'. The table lists various fruits and their prices and stock quantities. At the bottom right, there is a green circular button with a white plus sign. The status bar at the top shows the time as 1:55 and various icons.

ITEM	PRICE	QUANTITY IN STOCK
Apples	\$43.00	54
Bananas	\$1.00	21
Bluberry	\$43.00	0
Honey	\$4.00	23
Raspberry	\$6.00	92
Strawberry	\$5.00	5
Tomatoes	\$5.00	32

## تغییر دادن عنصر

```
private fun editItem() {
    val action = ItemDetailFragmentDirections.actionItemDetailFragmentToAddItemFragment(
        getString(R.string.edit_fragment_title),
        item.id
    )
    this.findNavController().navigate(action)
}
```

```
private fun bind(item: Item) {
    val price = "%.2f".format(item.itemPrice)
    binding.apply {
        itemName.setText(item.itemName, TextView.BufferType.SPANNABLE)
        itemPrice.setText(price, TextView.BufferType.SPANNABLE)
        itemCount.setText(item.quantityInStock.toString(), TextView.BufferType.SPANNABLE)
    }
}
```

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    val id = navigationArgs.itemId
    if (id > 0) {
        viewModel.retrieveItem(id).observe(this.viewLifecycleOwner) { selectedItem ->
            item = selectedItem
            bind(item)
        }
    } else {
        binding.saveAction.setOnClickListener {
            addNewItem()
        }
    }
}
```

```
private fun getUpdatedItemEntry(
    itemId: Int,
    itemName: String,
    itemPrice: String,
    itemCount: String
): Item {
    return Item(
        id = itemId,
        itemName = itemName,
        itemPrice = itemPrice.toDouble(),
        quantityInStock = itemCount.toInt()
    )
}
```

## ادامہ تغییر

```
fun updateItem(  
    itemId: Int,  
    itemName: String,  
    itemPrice: String,  
    itemCount: String  
) {  
    val updatedItem = getUpdatedItemEntry(itemId, itemName, itemPrice, itemCount)  
    updateItem(updatedItem)  
}  
  
private fun updateItem() {  
    if (isEntryValid()) {  
        viewModel.updateItem(  
            this.navigationArgs.itemId,  
            this.binding.itemName.text.toString(),  
            this.binding.itemPrice.text.toString(),  
            this.binding.itemCount.text.toString()  
        )  
        val action = AddItemFragmentDirections.actionAddItemFragmentToItemListFragment()  
        findNavController().navigate(action)  
    }  
}
```

# نسخه نهایی برنامه برای تغییر

Item Details


Strawberry

\$5.00

Quantity in Stock: 5

SELL

DELETE



Edit Item

Item Name \*  
Strawberry

Item Price \*  
\$ 5.00

Quantity in Stock \*  
5

SAVE

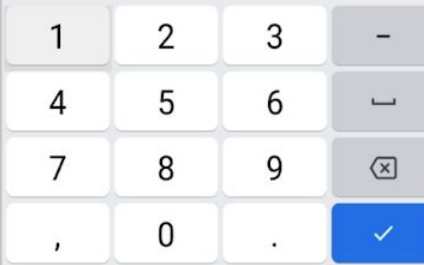
Edit Item

Item Name \*  
Strawberry

Item Price \*  
\$ 5.00

Quantity in Stock \*  
36

SAVE



Inventory

ITEM	PRICE	QUANTITY IN STOCK
Apples	\$43.00	54
Bananas	\$1.00	21
Blueberry	\$43.00	0
Honey	\$4.00	23
Raspberry	\$6.00	92
Strawberry	\$5.00	36
Tomatoes	\$5.00	32





سوال؟