DOCUMENTATION OF NjePrint APPLICATION

EMRECAN ERKUŞ

CONTENT OF DOCUMENTATION

Requirements and Usage Scenarios

Goal of this application

Our librarians have to calculate the prices of prints. But we have so much options(Colored A3 copy, colored A4 print,A4 scan,A3 scan, Black White A3 copy etc.).So in each of purchase, Our librarians have to check prices of selection and multiply with quantity and if customer want to use other options, Librarians have to check price again and again. So this application will make this process easier. The prices stored in database. So Librarians will just enter quantity then our application will calculate. As you know prices of prints can’t be stable. So with update part Librarian can update prices of prints.

Requirements

This application written by native language. That means this application only runs in Android devices.

Our minimum SDK for this application is 24.That means this application is suitable for Android 7.0 version and above.

That means this application can run in 96.19% of android devices.

Usage Scenarios

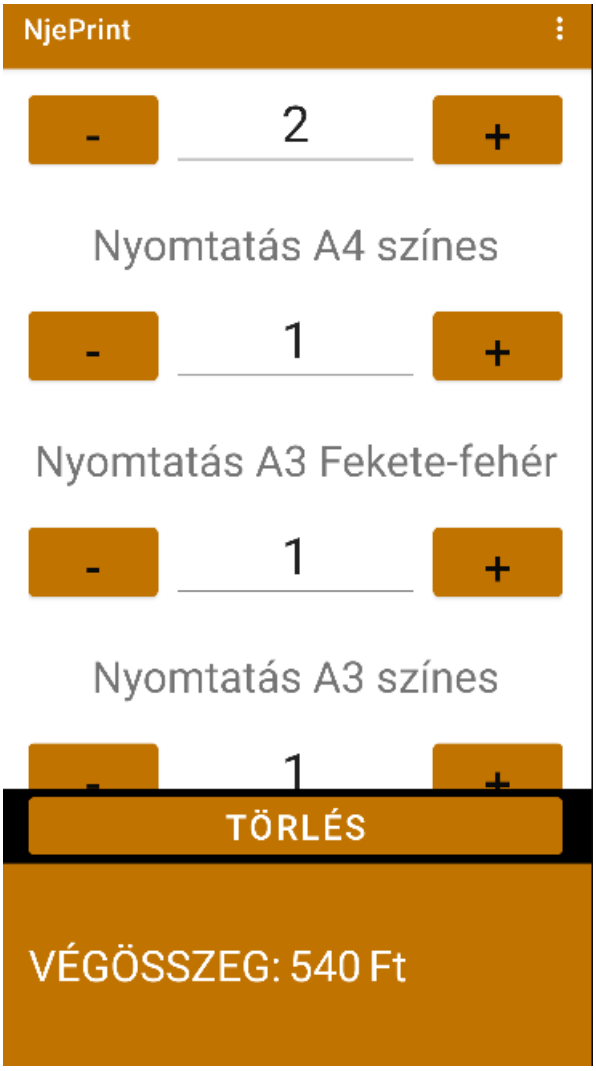
When you run the application, Splash screen welcomes user.

NjePrint

After that user will see user interface of application.

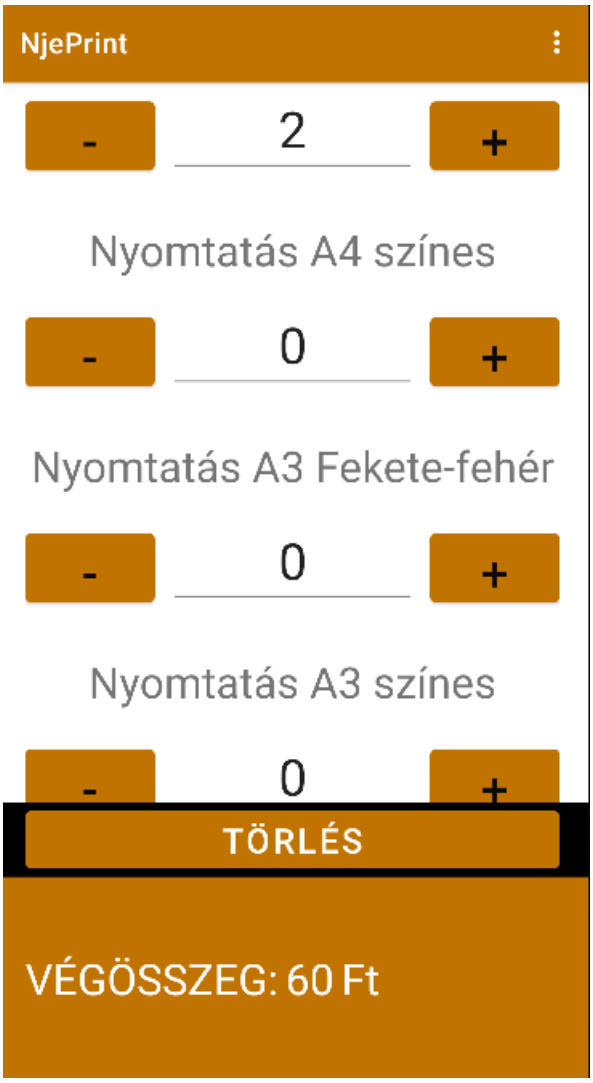
When user click to plus button, it will increase one by one and Total value will be calculated

at the same time. So user won’t need to click any button for calculate. It runs simultaneously. These prices will be calculated according to data which come from database.



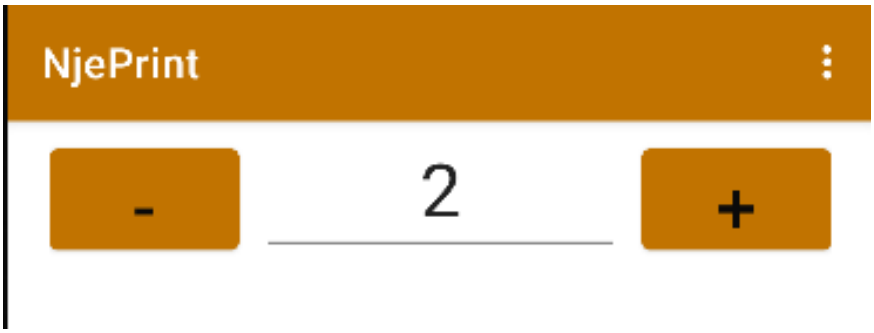
When user click to minus button, it will decrease one by one and Total value will be

calculated at the same time. So user won’t need to click any button for calculate. It runs simultaneously like plus button. These prices will be calculated according to data which come from database.



I built Recycler View for UI. So user can scroll down.

As you know prices of prints are changeable. Maybe sometimes user needs to update prices. So I built menu. When user click to menu, user will have only one option. It is update prices option.



When user click update prices menu, User will can update prices of database. So calculator will calculate according to updated prices.

I built Recycler View for this updating page. So user can scroll down.

When user want to clear all quantity and total value, user can click clear button. It will reset all quantities and total value.

As you know sometimes customers can want to print so many document. For this purpose plus button is not convenient. For example customer wants 200 A4 copy page. User has to click 200 times to plus button. It sounds so useless. But user can enter prices manually in this application and it runs the same as plus or minus button.

CODE REVIEW

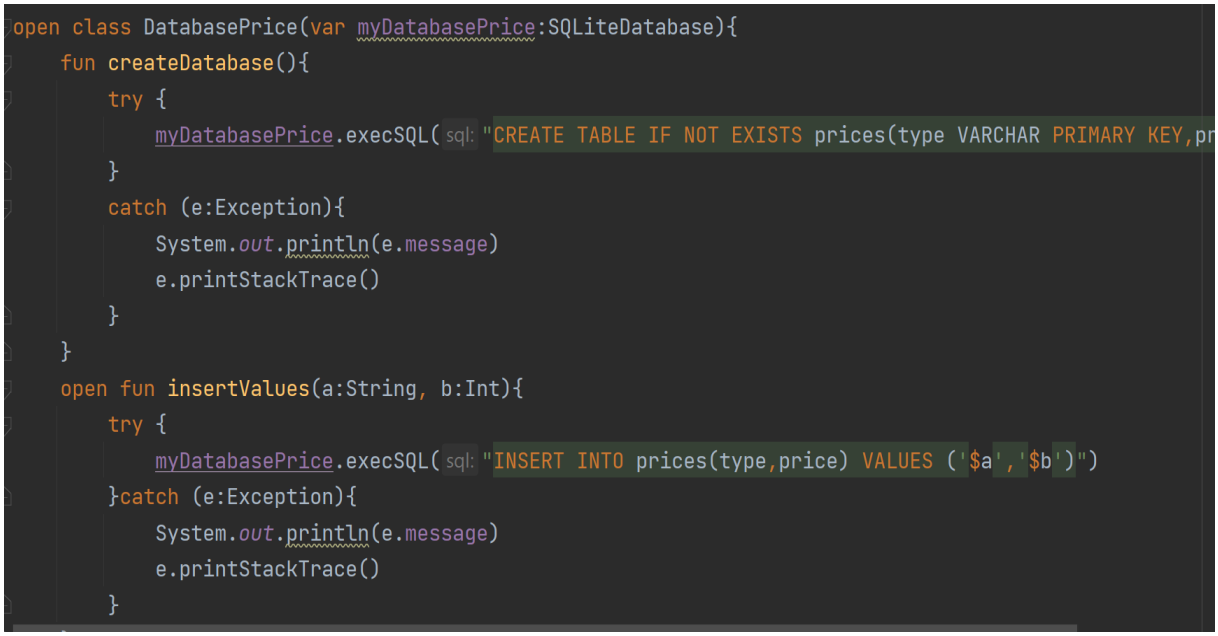
This is our data model that will be calculated and stored in database.

```
package com.example.njeprint.datamodel

class Data(var type:String,var price:Int) {
}
```

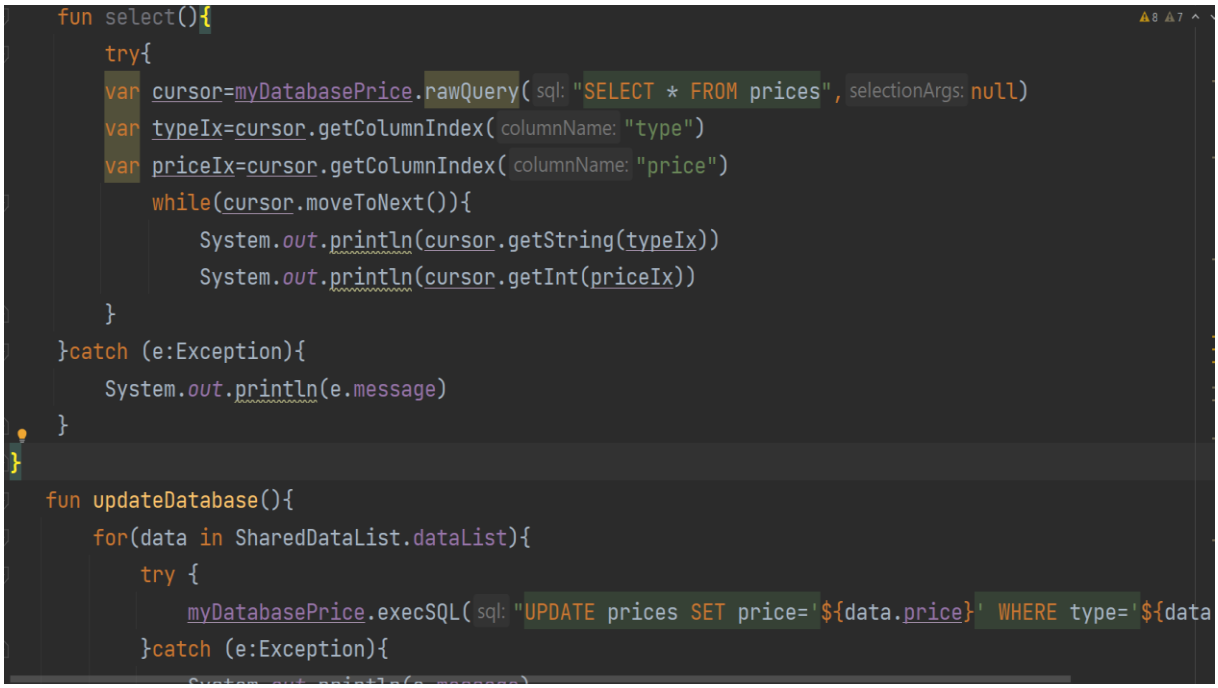
This is our data list that will be stored and used for Recycler View. It is singleton. Because we will use this structure for not only one activity.

```
package com.example.njeprint.datamodel

import com.example.njeprint.datamodel.Data

object SharedDataList {
    val dataList=ArrayList<Data>()
}
```

I built database class. So you don’t have to write SQL commands(CREATE,DELETE,UPDATE,INSERT) repeatedly.

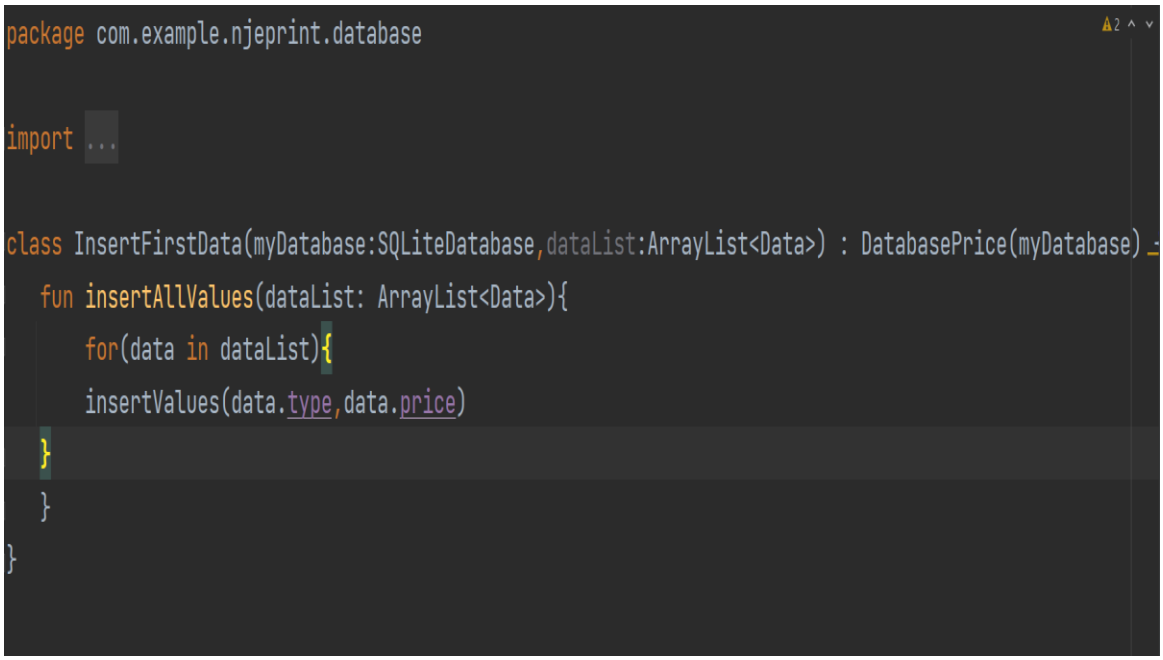


```
open class DatabasePrice(var myDatabasePrice:SQLiteDatabase){
    fun createDatabase(){
        try {
            myDatabasePrice.execSQL( sql: "CREATE TABLE IF NOT EXISTS prices(type VARCHAR PRIMARY KEY,pr
        }
        catch (e:Exception){
            System.out.println(e.message)
            e.printStackTrace()
        }
    }
    open fun insertValues(a:String, b:Int){
        try {
            myDatabasePrice.execSQL( sql: "INSERT INTO prices(type,price) VALUES ('$a','$b')")
        }catch (e:Exception){
            System.out.println(e.message)
            e.printStackTrace()
        }
```



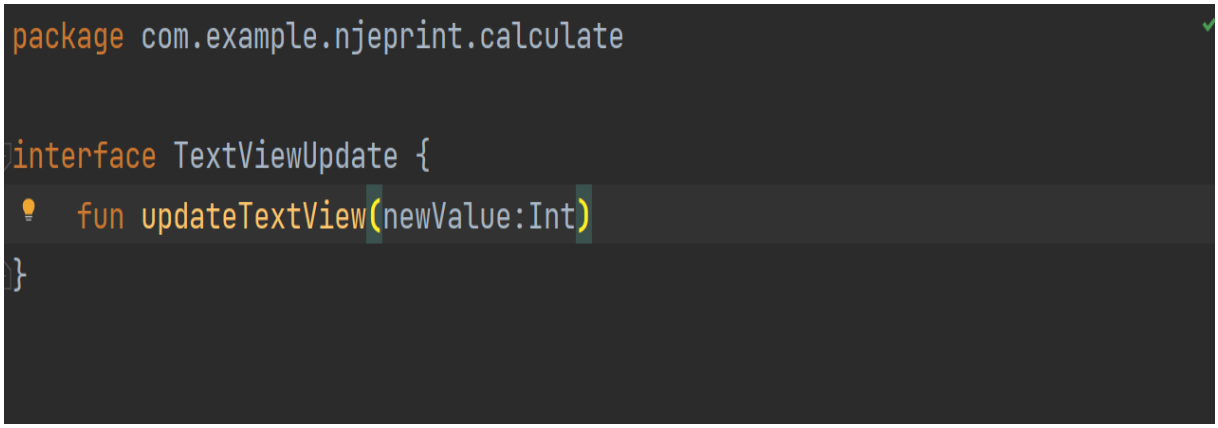
```
    fun select(){
        try{
        var cursor=myDatabasePrice.rawQuery( sql: "SELECT * FROM prices", selectionArgs: null)
        var typeIx=cursor.getColumnIndex( columnName: "type")
        var priceIx=cursor.getColumnIndex( columnName: "price")
            while(cursor.moveToNext()){
                System.out.println(cursor.getString(typeIx))
                System.out.println(cursor.getInt(priceIx))
        }
    }catch (e:Exception){
        System.out.println(e.message)
    }
}
    fun updateDatabase(){
        for(data in SharedDataList.dataList){
            try {
                myDatabasePrice.execSQL( sql: "UPDATE prices SET price='${data.price}' WHERE type='${data
            }catch (e:Exception){
```

I built a class that can insert first prices for database. For insert we have insert function from Database Price class. So that’s why I got inheritance from Database Price class.



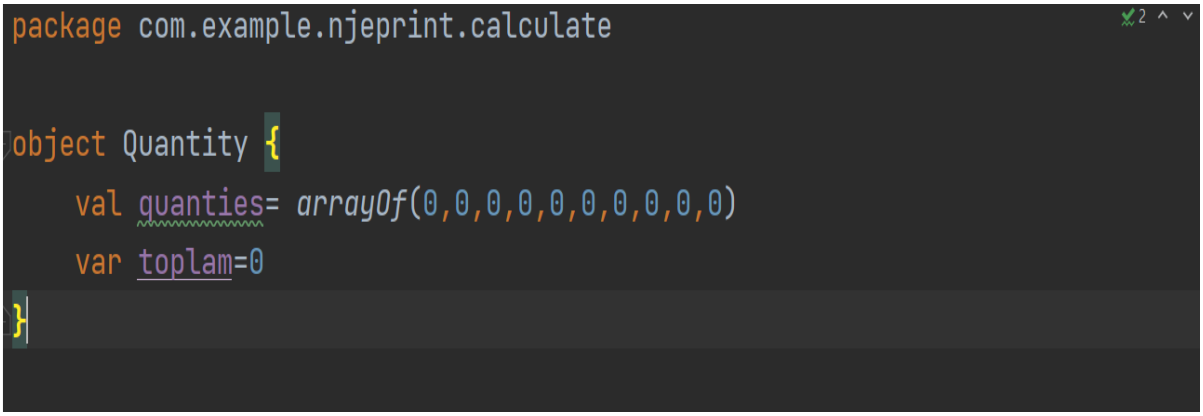
```
package com.example.njeprint.database

import ...

class InsertFirstData(myDatabase:SQLiteDatabase,dataList:ArrayList<Data>) : DatabasePrice(myDatabase) {
    fun insertAllValues(dataList: ArrayList<Data>){
        for(data in dataList){
        insertValues(data.type,data.price)
    }
    }
}
```

I built interface for calculate and update total price from Recycler View to Main Activity.



```
package com.example.njeprint.calculate

interface TextViewUpdate {
    fun updateTextView(newValue:Int)
}
```

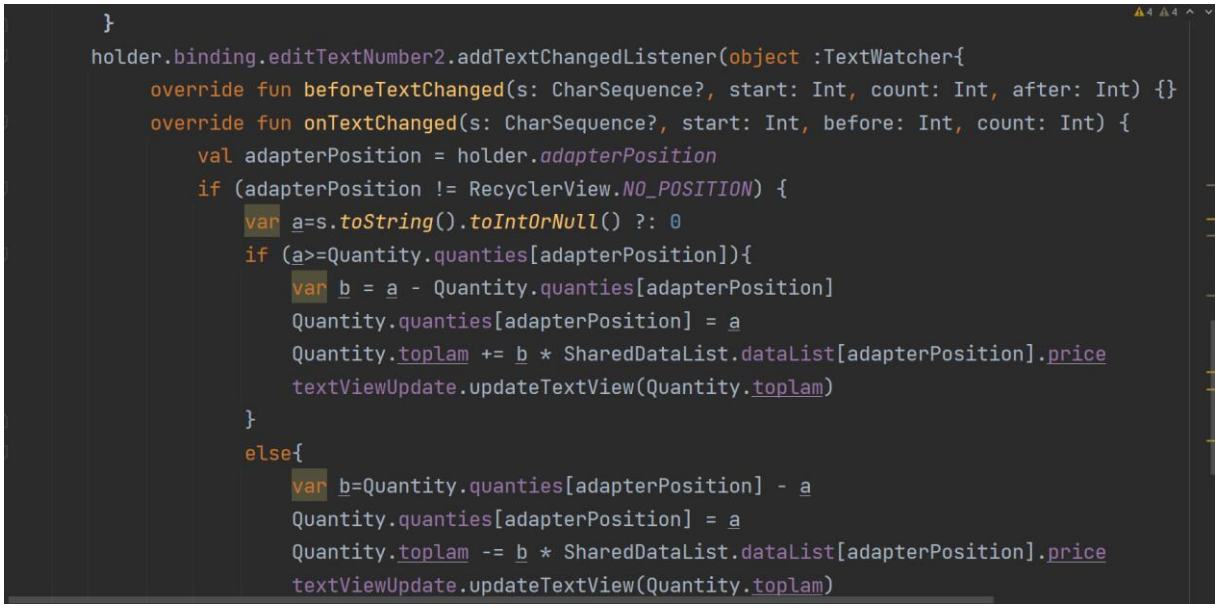
I built Quantity singleton for calculate, reset and update quantities from everywhere.



```
package com.example.njeprint.calculate

object Quantity {
    val quanties= arrayOf(0,0,0,0,0,0,0,0,0,0)
    var toplam=0
}
```

I built Recycler View adapter class for binding recycler view to main activity.

```
override fun onBindViewHolder(holder: MainRecHolder, position: Int) {
    val stringArray=holder.itemView.context.resources.getStringArray(R.array.names)
    holder.binding.text.text=stringArray.get(position)
    holder.binding.editTextNumber2.setText(Quantity.quanties[position].toString())
    holder.binding.buttonAdd.setOnClickListener { it: View!
        Quantity.quanties[position]=Quantity.quanties[position] + 1
        var a=Quantity.quanties[position].toString()
        holder.binding.editTextNumber2.setText(a)
        Quantity.toplam=Quantity.toplam + SharedDataList.dataList.get(position).price
        textViewUpdate.updateTextView(Quantity.toplam)
    }
    holder.binding.buttonMinus.setOnClickListener { it: View!
       if(Quantity.quanties[position]>0){
           Quantity.quanties[position]=Quantity.quanties[position]-1
           holder.binding.editTextNumber2.setText(Quantity.quanties[position].toString())
           Quantity.toplam=Quantity.toplam - SharedDataList.dataList.get(position).price
           textViewUpdate.updateTextView(Quantity.toplam)
       }
```



```
    }
    holder.binding.editTextNumber2.addTextChangedListener(object :TextWatcher{
        override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {}
        override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {
            val adapterPosition = holder.adapterPosition
            if (adapterPosition != RecyclerView.NO_POSITION) {
                var a=s.toString().toIntOrNull() ?: 0
                if (a>=Quantity.quanties[adapterPosition]){
                    var b = a - Quantity.quanties[adapterPosition]
                    Quantity.quanties[adapterPosition] = a
                    Quantity.toplam += b * SharedDataList.dataList[adapterPosition].price
                    textViewUpdate.updateTextView(Quantity.toplam)
                }
                else{
                    var b=Quantity.quanties[adapterPosition] - a
                    Quantity.quanties[adapterPosition] = a
                    Quantity.toplam -= b * SharedDataList.dataList[adapterPosition].price
                    textViewUpdate.updateTextView(Quantity.toplam)
```

I built Recycler View adapter class for binding recycler view to update activity.

```
package com.example.njeprint.adapter

import ...

class RecAdapter(val dataList:ArrayList<Data>) :RecyclerView.Adapter<RecAdapter.RecHolder>(){
class RecHolder(val binding:RecycleBinding):RecyclerView.ViewHolder(binding.root){

}

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecHolder {
        val binding=RecycleBinding.inflate(LayoutInflater.from(parent.context),parent, attachToParent: fals
        return RecHolder(binding)
    }

    override fun getItemCount(): Int {
        return dataList.size
    }
```

```
    override fun getItemCount(): Int {
        return dataList.size
    }

    override fun onBindViewHolder(holder: RecHolder, position: Int) {
        val stringArray=holder.itemView.context.resources.getStringArray(R.array.names)//Take from str
        holder.binding.typeText.text=stringArray[position]
        holder.binding.editTextNumber.hint=dataList.get(position).price.toString()
        holder.binding.editTextNumber.addTextChangedListener(object : TextWatcher {
            override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {}
            override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {
                val adapterPosition = holder.adapterPosition
                if (adapterPosition != RecyclerView.NO_POSITION) {
                    dataList[adapterPosition].price = s.toString().toInt()
                }
            }
            override fun afterTextChanged(s: Editable?) {}
        })
```

DEPENDECY AND SETUP

DEPENDECY

As I mentioned previous pages, our application was written by native language.

That means our application only runs in Android devices. Our application runs in Android devices which has 7.0 version and higher.

SETUP

We will create Microsoft teams chat group that contains librarians. We will send our apk version of application via this group. Then librarians will can download our application.

PERFORMANCE AND SAFETY

PERFORMANCE

The size of application is about 5MB. It is not much for storage of phone. It is so small size for performance. Performance is not affacted by our application.

SAFETY

We won’t publish our application to everyone. We will publish our application via Microsoft Teams to librarians. And also I didn’t use server or etc for database. I used local database of phone. So application safety is related to phone safety of librarians. Data will be stored in database of phones.

VERSION NOTES

It runs in current version of android devices. If we have any issue related with future versions, I will update to document and inform you.

Github URL of NjeQR codes for devolopers:

https://github.com/Emrecan-and/NjePrint