Министерство образования и молодежной политики Свердловской области



ГАПОУ СО «Екатеринбургский колледж транспортного строительства»

Отчёт по программе «**Неявные интенты**»

Выполнил: Полеев Андрей

Группа: ПР-31

Преподаватель: Мирошниченко Г.В

2023

**Входные данные:**

title(String) – название преступления

**Выходные данные:**

**recyclerView –** список

**Toast – сообщение**

**Листинг программы:**

@Entity  
data class Crime(@PrimaryKey var id: UUID = UUID.randomUUID())  
{  
 var title: String = ""  
 var date: Date = Date()  
 var isSolved: Boolean = false  
 var requiresPolice: Boolean = false  
 var suspect: String=""  
  
}

@Dao  
interface CrimeDao {  
 @Query("SELECT \* FROM crime")  
 fun getCrimes(): LiveData<List<Crime>>  
 @Query("SELECT \* FROM crime WHERE id=(:id)")  
 fun getCrime(id: UUID): LiveData<Crime?>  
 @Update  
 fun updateCrime(crime: Crime)  
 @Insert  
 fun addCrime(crime: Crime)  
 @Query("SELECT \* FROM crime WHERE title = :title")  
 suspend fun getCrimeByTitle(title: String): Crime?  
}

@Database(entities = [ Crime::class ],  
 version=1)  
@TypeConverters(CrimeTypeConverters::class)  
abstract class CrimeDatabase : RoomDatabase() {  
 abstract fun crimeDao(): CrimeDao  
}  
val *migration\_1\_2* = object : Migration(1,2){  
 override fun migrate(database: SupportSQLiteDatabase) {  
 database.execSQL("ALTER TABLE Crime ADD COLUMN suspect TEXT NOT NULL DEFAULT ''")  
 }  
}

class CrimeDatailModel(): ViewModel() {  
 private val repository=CrimeRepository.get()  
  
 suspend fun getCrimeByTitle(title: String): Crime? {  
 return repository.getCrimeByTitle(title)  
 }  
  
 fun save(crime: Crime) {  
 repository.updateCrime(crime)  
 }  
 fun add(crime: Crime){  
 repository.addCrime(crime)  
 }  
}

class CrimeFragment : Fragment() {  
  
 private lateinit var crime: Crime  
 private lateinit var title: EditText  
 private lateinit var dateButton: Button  
 private lateinit var solvedCheckBox: CheckBox  
 private lateinit var reportButton: Button  
 private lateinit var suspectButton: Button  
 private lateinit var listButton: Button  
 private lateinit var addButoon: Button  
 private val crimeDetailViewModel:CrimeDatailModel by *lazy***{** ViewModelProviders.of(this).get(CrimeDatailModel::class.*java*)  
 **}** override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 crime = Crime()  
 }  
  
 override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?): View? {  
 val view = inflater.inflate(R.layout.*fragment\_crime*, container, false)  
 title = view.findViewById(R.id.*crime\_title*) as EditText  
 dateButton = view.findViewById(R.id.*crime\_date*) as Button  
 solvedCheckBox = view.findViewById(R.id.*crime\_solved*) as CheckBox  
 reportButton=view.findViewById(R.id.*crime\_report*) as Button  
 suspectButton=view.findViewById(R.id.*crime\_suspect*) as Button  
// listButton = view.findViewById(R.id.listButton) as Button  
 addButoon = view.findViewById(R.id.*addButton*) as Button  
  
 dateButton.*apply* **{** *text* = crime.date.toString(); *isEnabled* = false **}** return view  
 }  
  
  
 override fun onStart() {  
 super.onStart()  
 val titleWatcher = object : TextWatcher {  
 override fun beforeTextChanged(sequence: CharSequence?, start: Int, count: Int,  
 after: Int) {}  
  
 override fun onTextChanged(sequence: CharSequence?, start: Int, before: Int,  
 count: Int) {  
 crime.title = sequence.*toString*()  
 }  
  
 override fun afterTextChanged(sequence: Editable?) {}  
 }  
 solvedCheckBox.setOnClickListener()**{** suspectButton.*isEnabled* = solvedCheckBox.*isChecked* reportButton.*isEnabled* = solvedCheckBox.*isChecked* **}** title.addTextChangedListener(titleWatcher)  
 solvedCheckBox.*apply* **{** setOnCheckedChangeListener **{** \_, isChecked **->** crime.isSolved = isChecked  
 **}  
 }** reportButton.setOnClickListener **{** if(title.*text*.*isNotEmpty*()) {  
 Intent(Intent.*ACTION\_SEND*).*apply* **{** *type* = "text/plain"  
 putExtra(Intent.*EXTRA\_TEXT*, getCrimeReport())  
 putExtra(Intent.*EXTRA\_SUBJECT*, getString(R.string.*crime\_report\_subject*))  
 **}**.*also* **{** intent **->** val chooserIntent =  
 Intent.createChooser(intent, getString(R.string.*send\_report*))  
 startActivity(chooserIntent)  
 **}** }else{  
 Toast.makeText(*context*,"Напишите название преступления",Toast.*LENGTH\_SHORT*).show()  
 }  
  
 **}** suspectButton.*apply***{** val pickContactIntent =  
 Intent(Intent.*ACTION\_PICK*, ContactsContract.Contacts.*CONTENT\_URI*)  
 setOnClickListener **{** startActivityForResult(pickContactIntent, *REQUEST\_CONTACT*)  
 **}** val packageManager: PackageManager = requireActivity().*packageManager* val resolvedActivity: ResolveInfo? = packageManager.resolveActivity(  
 pickContactIntent,  
 PackageManager.*MATCH\_DEFAULT\_ONLY* )  
 if (resolvedActivity == null) {  
 *isEnabled* = false  
 }  
  
 **}** addButoon.setOnClickListener()**{** var existingCrime:Crime?  
  
 if (title.*text*.*isEmpty*()) {  
 Toast.makeText(requireActivity(), "Введите преступление", Toast.*LENGTH\_SHORT*).show()  
 } else {  
 val crimeTitle = title.*text*.toString()  
  
  
  
 GlobalScope.*launch*(Dispatchers.IO) **{** existingCrime = crimeDetailViewModel.getCrimeByTitle(title.*text*.toString())  
 *CoroutineScope*(Dispatchers.Main).*launch* **{** if (existingCrime == null) {  
  
 val crime = Crime()  
 crime.title = crimeTitle  
 crime.date = Date()  
 crime.isSolved = solvedCheckBox.*isChecked* crimeDetailViewModel.add(crime)  
  
 title.*text* = null  
 Toast.makeText(  
 requireActivity(),  
 "Вы добавили новое преступление",  
 Toast.*LENGTH\_SHORT* ).show()  
 } else {  
 Toast.makeText(  
 requireActivity(),  
 "В списке уже есть такое название",  
 Toast.*LENGTH\_SHORT* ).show()  
 }  
 **}  
  
  
 }** }  
 **}** }  
  
 private fun updateUI(){  
 title.setText(crime.title)  
 solvedCheckBox.*apply* **{** *isChecked*=crime.isSolved!!  
 jumpDrawablesToCurrentState()  
 **}** if (crime.suspect.*isNotEmpty*()){  
 suspectButton.*text*=crime.suspect  
 }  
 }  
 private fun getCrimeReport(): String{  
 val solvedString = if (crime.isSolved == true){  
 getString(R.string.*crime\_report\_solved*)  
 }  
 else{  
 getString(R.string.*crime\_report\_unsolved*)  
 }  
 val dateString= DateFormat.format(*DATE\_FORMAT*,crime.date).toString()  
 var suspect=if (crime.suspect.*isBlank*()){  
 getString(R.string.*crime\_report\_no\_suspect*)  
 }  
 else{  
 getString(R.string.*crime\_report\_suspect*, crime.suspect)  
 }  
 return getString(R.string.*crime\_report*,crime.title,dateString,solvedString,suspect)  
 }  
 @RequiresApi(Build.VERSION\_CODES.*O*)  
 override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {  
 when{  
 resultCode!= Activity.*RESULT\_OK* -> return  
 requestCode== *REQUEST\_CONTACT* && data !=null ->{  
 val contactUri: Uri? =data.*data* val queryFields=*arrayOf*(ContactsContract.Contacts.*DISPLAY\_NAME*)  
 val cursor =  
 contactUri?.*let* **{** requireActivity().*contentResolver*.query(**it**,queryFields,null,  
 null)  
 **}** cursor?.*use* **{** if (**it**.*count*==0){  
 return  
 }  
 **it**.moveToFirst()  
 val suspect=**it**.getString(0)  
 crime.suspect=suspect  
 crimeDetailViewModel.save(crime)  
 suspectButton.*text*=suspect  
 **}** }  
 }  
 }  
 companion object{  
 fun newInstance() : CrimeListFragment{  
 return CrimeListFragment()  
 }  
 }  
}

class CrimeListFragment : Fragment() {  
  
 private val common = 0  
 private val serious = 3  
 private lateinit var recyclerView: RecyclerView  
 private var adapter: CrimeAdapter? = null  
 private val crimeListViewModel: CrimeListViewModel by *lazy* **{** ViewModelProviders.of(this).get(CrimeListViewModel::class.*java*)  
 **}** override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setHasOptionsMenu(true)  
 }  
  
 override fun onCreateView(  
 inflater: LayoutInflater,  
 container: ViewGroup?,  
 savedInstanceState: Bundle?): View? {  
 val view = inflater.inflate(R.layout.*fragment\_crime\_list*, container, false)  
 recyclerView = view.findViewById(R.id.*crime\_recycler\_view*) as RecyclerView  
 recyclerView.*layoutManager* = LinearLayoutManager(*context*)  
 recyclerView.*adapter*=adapter  
 return view  
 }  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
 crimeListViewModel.crimeListLiveData?.observe(  
 *viewLifecycleOwner*,  
 androidx.lifecycle.*Observer* **{** crimes**->**crimes.*let* **{** Log.i(*TAG*,"Количество: ${crimes.size}")  
 updateUI(crimes) **}  
 }** )  
 }  
 private fun updateUI(crimes:List<Crime>){  
 adapter=CrimeAdapter(crimes)  
 recyclerView.*adapter*=adapter  
 }  
  
 companion object {  
 fun newInstance(): CrimeListFragment {  
 return CrimeListFragment()  
 }  
 }  
  
 private inner class CrimeAdapter(var crimes: List<Crime>):RecyclerView.Adapter<CrimeHolder>(){  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):CrimeHolder {  
 val view=*layoutInflater*.inflate(R.layout.*list\_item\_common\_crime*,parent,false)  
 return CrimeHolder(view)  
 }  
  
  
 override fun onBindViewHolder(holder: CrimeHolder, position: Int) {  
  
 val crime=crimes[position]  
 holder.bind(crime)  
 }  
  
 override fun getItemCount(): Int {  
 return crimes.size  
 }  
 }  
 private inner class CrimeHolder(view: View):RecyclerView.ViewHolder(view), View.OnClickListener {  
 private lateinit var crime: Crime  
 val titleTextView: TextView =itemView.findViewById(R.id.*crime\_title*)  
 val dateTextView: TextView = itemView.findViewById(R.id.*crime\_date*)  
 init {  
 itemView.setOnClickListener(this)  
 }  
 fun bind(crime:Crime){  
 this.crime=crime  
 titleTextView.*text*=this.crime.title  
 dateTextView.*text*=this.crime.date.toString()  
 }  
 override fun onClick(v: View) {  
  
 }

class CrimeListViewModel : ViewModel() {  
  
 val crimeRepository=CrimeRepository.get()  
 val crimeListLiveData=crimeRepository.getCrimes()  
}

class CrimeRepository private constructor(context: Context) {  
 private val database: CrimeDatabase = Room.databaseBuilder(  
 context.*applicationContext*,  
 CrimeDatabase::class.*java*,  
 *DATABASE\_NAME* ).addMigrations(*migration\_1\_2*).build()  
 private val crimeDao = database.crimeDao()  
 private val executor = Executors.newSingleThreadExecutor()  
 fun getCrimes(): LiveData<List<Crime>>? = crimeDao.getCrimes()  
 fun getCrime(id: UUID): LiveData<Crime?>? = crimeDao.getCrime(id)  
 fun updateCrime(crime: Crime) {  
 executor.execute **{** crimeDao.getCrime(crime.id)  
 **}** }  
  
 suspend fun getCrimeByTitle(title: String): Crime? {  
 return withContext(Dispatchers.IO) **{** crimeDao.getCrimeByTitle(title)  
 **}** }  
  
 fun addCrime(crime: Crime) {  
 executor.execute **{** crimeDao.addCrime(crime)  
 **}** }  
  
 companion object {  
 private var INSTANCE: CrimeRepository? = null  
 fun initialize(context: Context) {  
 if (INSTANCE == null) {  
 INSTANCE = CrimeRepository(context)  
 }  
 }  
  
 fun get(): CrimeRepository {  
 return INSTANCE ?: throw IllegalStateException("CrimeRepository must be initialized")  
 }  
 }  
}

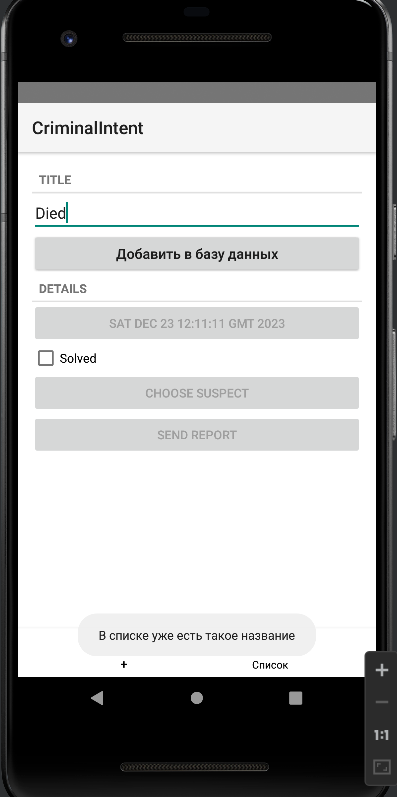
class CrimeTypeConverters {  
 @TypeConverter  
 fun fromDate(date: Date?): Long? {  
 return date?.*time* }  
  
 @TypeConverter  
 fun toDate(millisSinceEpoch: Long?): Date?  
 {  
 return millisSinceEpoch?.*let* **{** Date(**it**)  
 **}** }  
  
 @TypeConverter  
 fun toUUID(uuid: String?): UUID? {  
 return UUID.fromString(uuid)  
 }  
  
 @TypeConverter  
 fun fromUUID(uuid: UUID?): String? {  
 return uuid?.toString()  
 }  
}

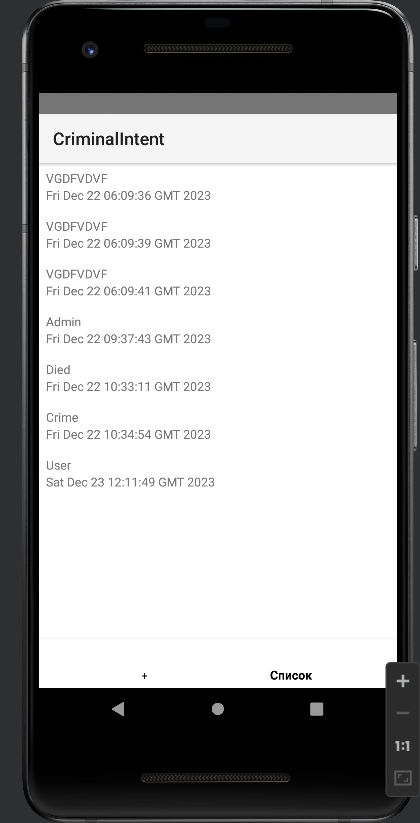
class CriminalIntentApplication : Application()  
{  
 override fun onCreate() {  
 super.onCreate()  
 CrimeRepository.initialize(this)  
 }  
}

class Fragment\_crime\_list : AppCompatActivity() {  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*fragment\_crime\_list*)  
 }  
}

class MainActivity : AppCompatActivity() {  
 private lateinit var nav: BottomNavigationView  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_main*)  
 nav = findViewById(R.id.*bottomNavigationView*)  
 nav.setOnNavigationItemSelectedListener **{** item **->** when (item.*itemId*) {  
 R.id.*addCrime* -> {  
 loadFragment(CrimeFragment())  
 return@setOnNavigationItemSelectedListener true  
 }  
 R.id.*listCrimes* -> {  
  
 loadFragment(CrimeListFragment())  
 return@setOnNavigationItemSelectedListener true  
 }  
 }  
 false  
 **}** loadFragment(CrimeFragment())  
  
  
 }  
 private fun loadFragment(fragment: Fragment) {  
 *supportFragmentManager*.beginTransaction()  
 .replace(R.id.*fragment\_container*, fragment)  
 .commit()  
 }  
}

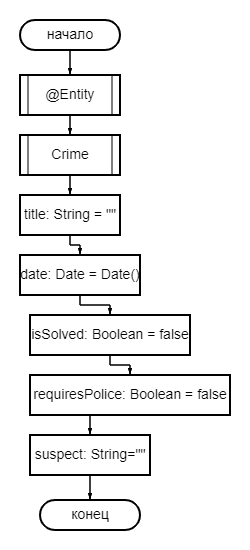
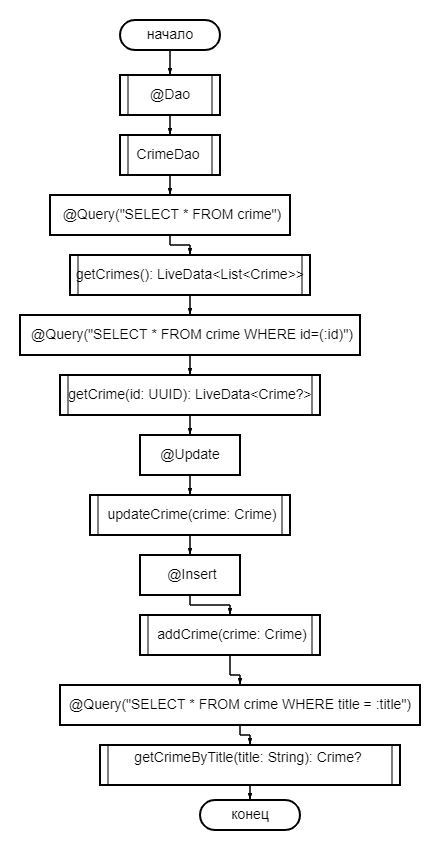
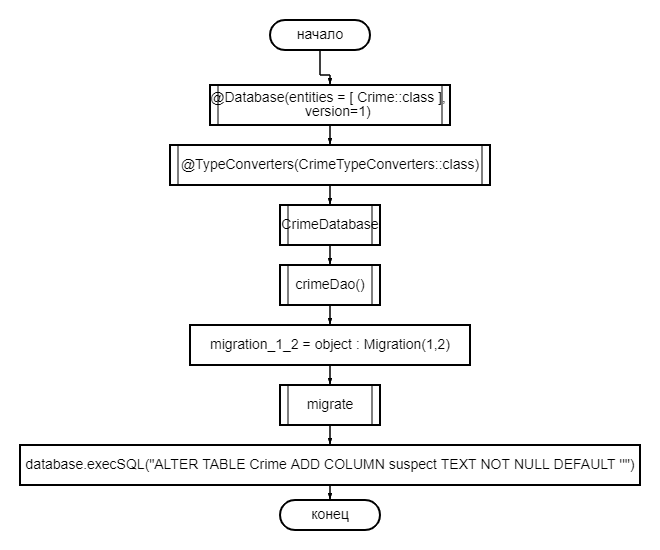
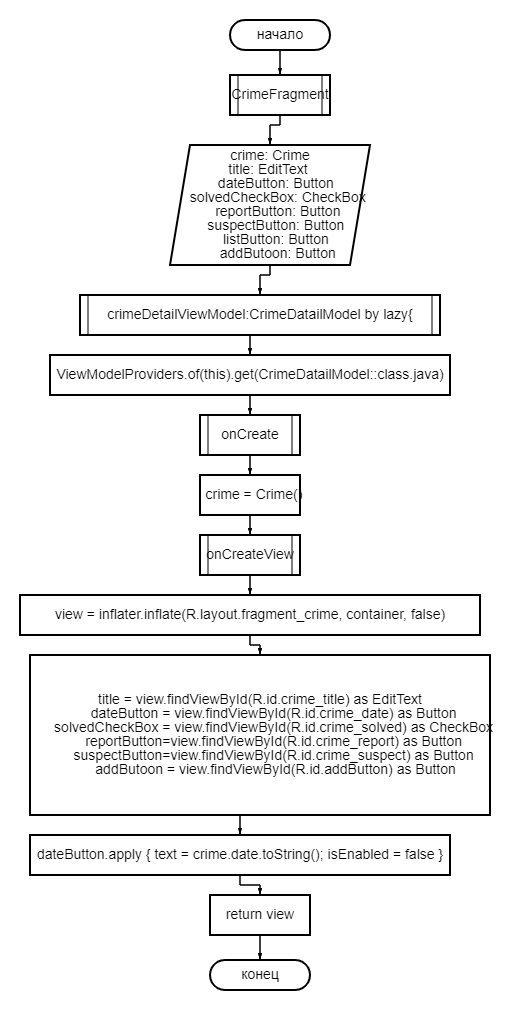
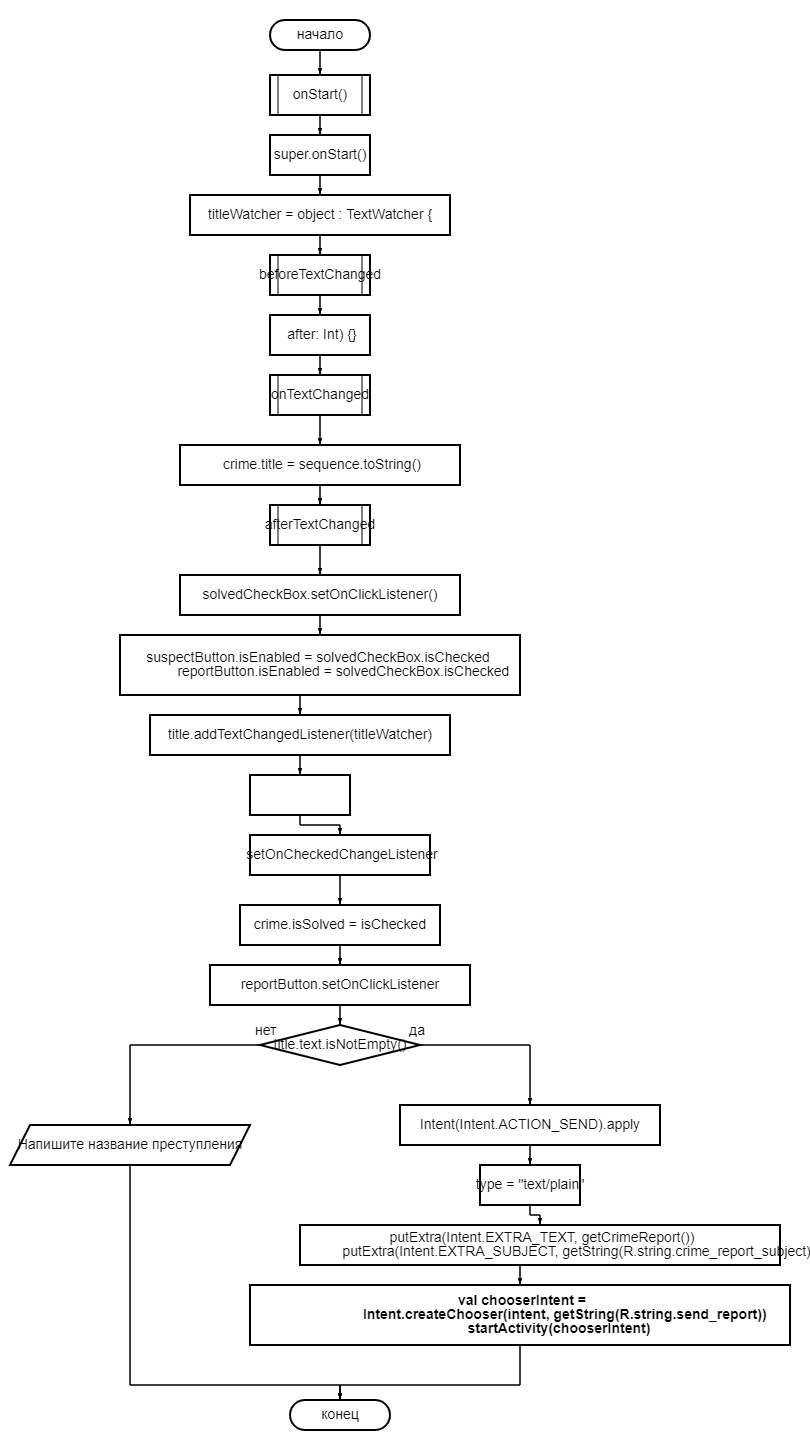
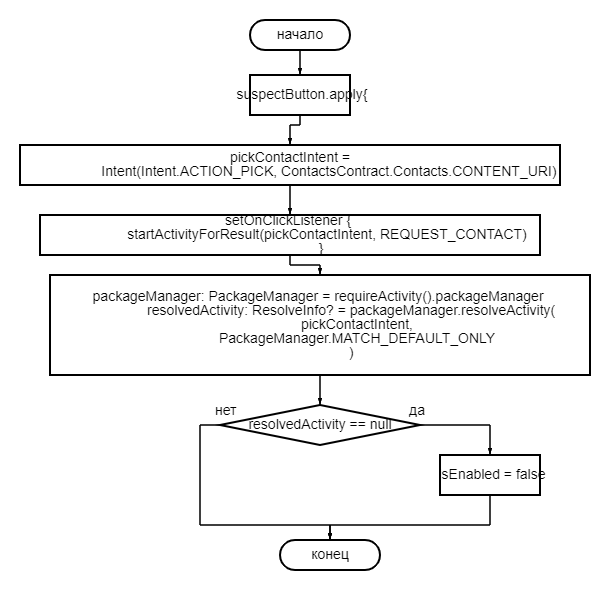
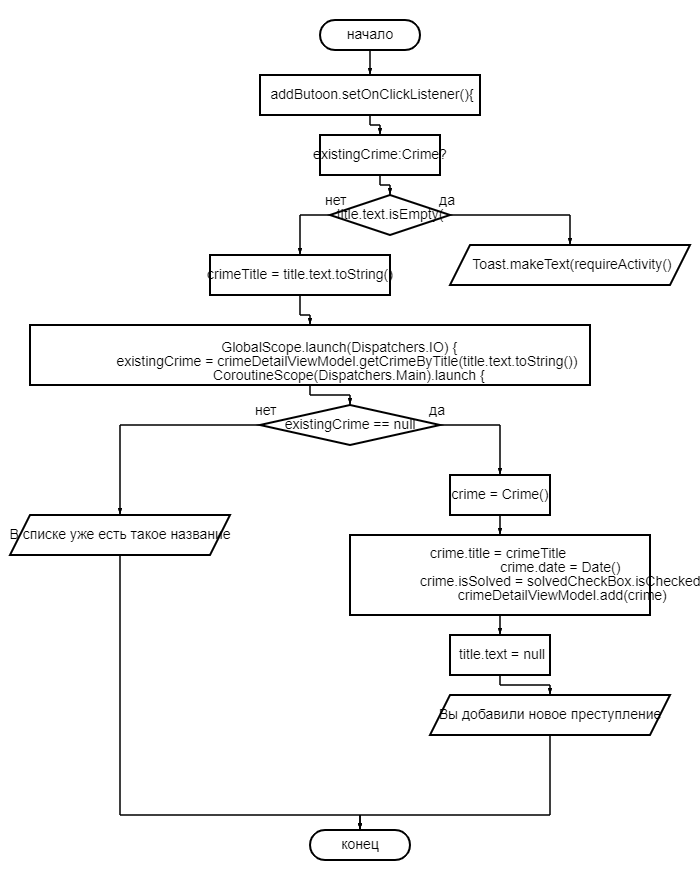
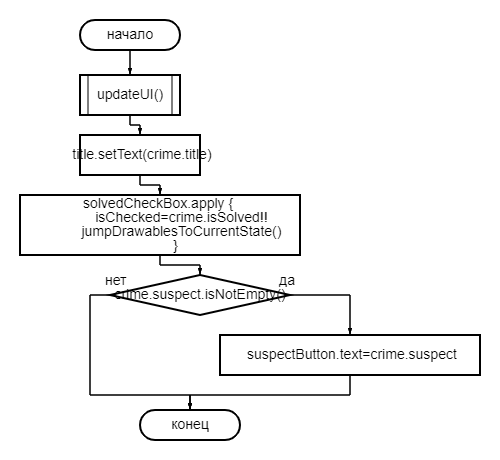
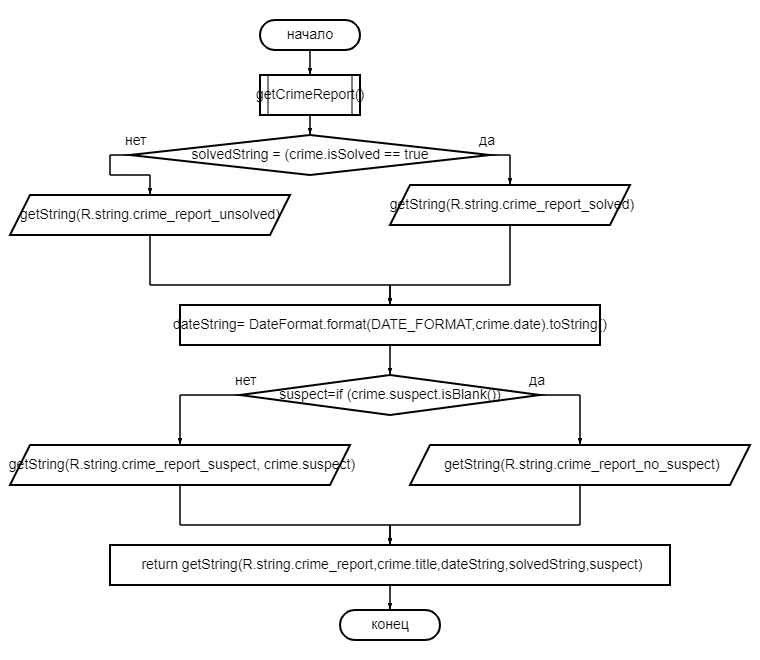
Приложение:





Блок-схема:







**Тестовые ситуации:**

**1)** Проверка на пустоту (if else)  
**2)** Проверка есть такие данные в списке (if else)  
**3)** Проверка на правильность ввода (if else)

**Вывод:** При выполнении данной практической работы я научился работать с неявными интентами.