

그림 １. ScanRemind 클래스 다이어그램

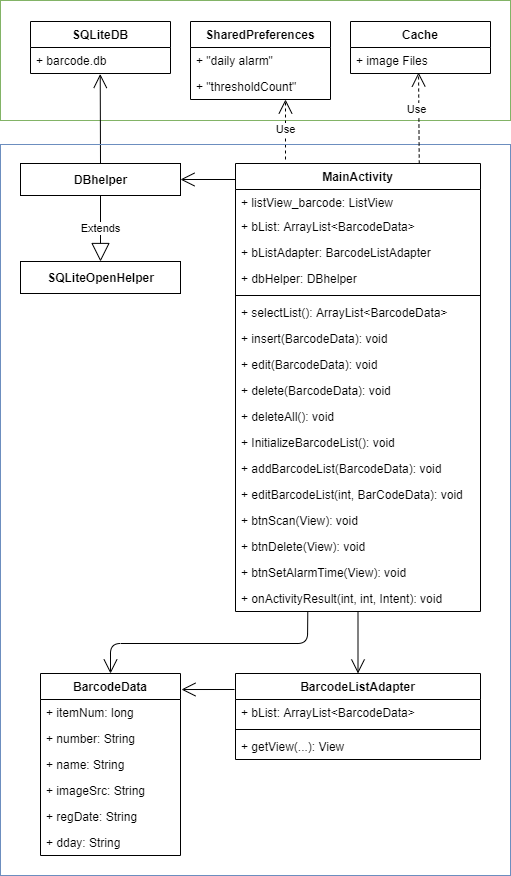


그림 ２. 메인 액티비티 클래스 다이어그램

메인 액티비티 클래스의 여러가지 동작 중 가장 대표적인 리스트에 항목을 추가하는 동작을 소스코드와 함께 설명한다. 수정, 삭제 동작도 같은 원리로 동작한다.

//추가 버튼 눌러서 SetScanInfoActivity로 이동  
public void btnScan(View v) {  
 Intent intent = new Intent(MainActivity.this, SetScanInfoActivity.class);  
 intent.putExtra("mode", "add");  
 startActivityForResult(intent, *REQUEST\_ADD\_ITEM*);  
 //startActivity(intent);  
}

메인 액티비티에서 추가 버튼을 누르면 btnScan()이 호출된다. btnScan()은 “mode” 인탠트를 “add”로 설정하고 SetScanInfoActivity를 호출하는 역할이다. “edit”을 설정하고 호출하면 SetScanInfoActivity가 수정 모드로 동작한다.

//스캔하고 메인액티비티로 넘어올 때 이벤트  
@Override  
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {  
 super.onActivityResult(requestCode, resultCode, data);  
 switch (requestCode) {  
 case *REQUEST\_ADD\_ITEM* : //SetScanInfoActivity's add result  
 if(resultCode == 1) {  
 addBarcodeList((BarcodeData) data.getSerializableExtra("barcodedata"));  
 } else {  
 //Toast.makeText(MainActivity.this, "add Failed", Toast.LENGTH\_SHORT).show();  
 }  
 break;  
 case *REQUEST\_EDIT\_ITEM* : //SetScanInfoActivity's edit result  
 if(resultCode == 1) {  
 editBarcodeList(data.getIntExtra("index", -1),  
 (BarcodeData) data.getSerializableExtra("barcodedata"));  
 } else {  
 //Toast.makeText(MainActivity.this, "edit Failed", Toast.LENGTH\_SHORT).show();  
 }  
 }  
}

SetScanInfoActivity가 정상적으로 종료되면 인탠트에서 바코드데이터를 얻어와 addBarcodeList()를 호출한다.

//바코드 리스트에 데이터 추가  
public void addBarcodeList(BarcodeData data) {  
 bList.add(data);  
 bListAdapter.notifyDataSetChanged();  
 insert(bList.get(bList.size() - 1));  
}

addBarcodeList()는 메인 액티비티의 멤버변수인 bList에 바코드 데이터를 추가하고 리스트뷰를 새로고침해 변동된 사항을 반영한다. 그리고 insert()를 호출해 SQLiteDB에 저장될 수 있도록 한다.

//DB에 바코드데이터 삽입  
private void insert(BarcodeData b) {  
 // Gets the data repository in write mode  
 SQLiteDatabase db = dbHelper.getWritableDatabase();  
 // Create a new map of values, where column names are the keys  
 ContentValues values = new ContentValues();  
 values.put(BarcodeDataContract.BarcodeTable.*COLUMN\_NAME\_ITEMNUM*, b.getItemNum());  
 values.put(BarcodeDataContract.BarcodeTable.*COLUMN\_NAME\_BNUMBER*, b.getNumber());  
 values.put(BarcodeDataContract.BarcodeTable.*COLUMN\_NAME\_BNAME*, b.getName());  
 values.put(BarcodeDataContract.BarcodeTable.*COLUMN\_NAME\_IMGSRC*, b.getImageSrc());  
 values.put(BarcodeDataContract.BarcodeTable.*COLUMN\_NAME\_REGDATE*, b.getRegDate());  
 values.put(BarcodeDataContract.BarcodeTable.*COLUMN\_NAME\_DDAY*, b.getDday());  
 // Insert the new row, returning the primary key value of the new row  
 long newRowId = db.insert(BarcodeDataContract.BarcodeTable.*TABLE\_NAME*, null, values);  
 Log.*d*("insert in DB", "data:" + b.toString());  
}

DB테이블에 바코드데이터를 삽입할 땐 dhHelper를 이용한다.

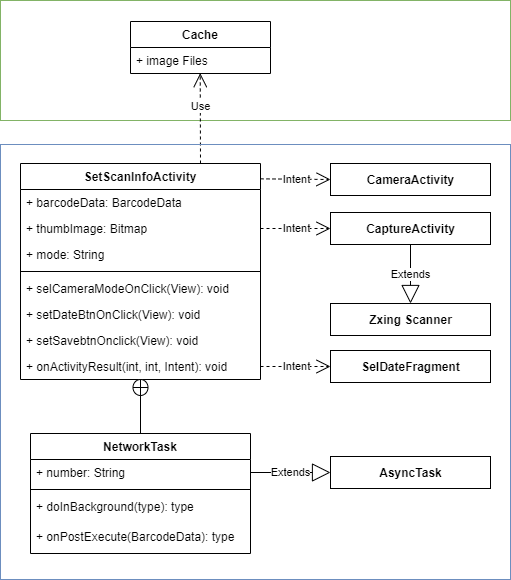


그림 ３. SetScanInfo 액티비티

SetScanInfoActivity의 동작 중 바코드를 검색해서 추가하는 기능을 소스코드와 함께 설명한다.

//메인액티비티에서 수정/추가 정보를 인텐트로 받아옴  
Intent intent = new Intent(this.getIntent());  
mode = intent.getStringExtra("mode");  
if(mode.equals("add")) {  
 saveButton.setText("저장");  
}

SetScanInfoActivity는 생성될 때 목적이 추가인지 수정인지 확인해서 mode를 설정한다.

//사진 촬영 클릭 이벤트  
public void selCameraModeOnClick(View view) {  
 int permissonCheck= ContextCompat.*checkSelfPermission*(this, Manifest.permission.*CAMERA*); //권한 확인  
 if(permissonCheck == PackageManager.*PERMISSION\_DENIED*) {  
 //Toast.makeText(getApplicationContext(), "카메라 권한 없음", Toast.LENGTH\_SHORT).show();  
 ActivityCompat.*requestPermissions*(this, new String[]{Manifest.permission.*CAMERA*}, 100);  
 }  
 if(permissonCheck == PackageManager.*PERMISSION\_GRANTED*) {  
 //인터넷 연결 되있으면  
 if(CheckNetworkStatus.*getConnectivityStatus*(getApplicationContext()) != CheckNetworkStatus.*TYPE\_NOT\_CONNECTED*) {  
 //바코드 촬영 or 일반 사진 촬영 중 택 1 다이얼로그 띄워야함  
 final AlertDialog.Builder alert = new AlertDialog.Builder(this);  
 View dlgView = getLayoutInflater().inflate(R.layout.*layout\_dialog\_camera\_sel*, null);  
 ImageView selCamera = dlgView.findViewById(R.id.*imageView\_camera*);  
 ImageView selBarcode = dlgView.findViewById(R.id.*imageView\_barcode*);  
 alert.setView(dlgView);  
  
 final AlertDialog alertDialog = alert.create();  
 alertDialog.setCanceledOnTouchOutside(true);

selCamera.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) { //일반 촬영을 선택한 경우  
 //일반 사진 촬영 엑티비티 호출  
 Intent takePictureIntent = new Intent(MediaStore.*ACTION\_IMAGE\_CAPTURE*);  
 if (takePictureIntent.resolveActivity(getPackageManager()) != null) {  
 startActivityForResult(takePictureIntent, *REQUEST\_IMAGE\_CAPTURE*);  
 }  
 alertDialog.dismiss();  
 }  
 });  
 selBarcode.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) { //바코드 촬영을 선택한 경우  
 //바코드 스캔하는 CaptureActivity로 이동  
 IntentIntegrator integrator = new IntentIntegrator(SetScanInfoActivity.this);  
 integrator.setCaptureActivity(CaptureActivity.class);  
 integrator.setOrientationLocked(false);  
 integrator.setDesiredBarcodeFormats(IntentIntegrator.*ALL\_CODE\_TYPES*);  
 integrator.setPrompt("SCAN CODE");  
 integrator.initiateScan(); //->onActivityResult case REQUEST\_BARCODE\_SCAN로 이동  
 alertDialog.dismiss();  
 }  
 });  
 alertDialog.show();  
 }  
 }  
}

사용자가 카메라 아이콘을 클릭하면 selCameraModeOnClick()가 호출된다. 가장 먼저 카메라 권한이 허용되있는지 확인한다. 그 다음 인터넷에 연결 가능한 상태인지 확인한다. 바코드 검색 기능을 이용하려면 코리안넷에 접속 가능해야하기 때문이다. 인터넷 접속이 가능하면 일반 촬영과 바코드 스캔을 선택할 수 있는 다이얼로그를 보여주고 선택한 방식의 촬영 엑티비티를 호출한다.

@Override  
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {  
 super.onActivityResult(requestCode, resultCode, data);  
 switch (requestCode) {  
 case *REQUEST\_BARCODE\_SCAN* : //바코드 스캔 결과  
 IntentResult result = IntentIntegrator.*parseActivityResult*(requestCode, resultCode, data);  
 if (result != null) {  
 if (result.getContents() != null) {  
 barcodeData.setNumber(result.getContents()); //데이터 객체 스캔된 바코드 넘버로 초기화  
 // AsyncTask를 통해 HttpURLConnection 수행.  
 NetworkTask networkTask = new NetworkTask(barcodeData.getNumber());  
 networkTask.execute();  
 }  
 else {  
 Toast.*makeText*(getApplicationContext(), "No Result", Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
 break;  
 }  
}

바코드 촬영을 선택한 경우 Zxing(Zebra Crossing) 라이브러리의 바코드 스캔 기능을 사용한다. 바코드 스캔이 성공하면 인텐트에 바코드 숫자가 담겨서 넘어오게 된다. SetScanInfoActivity는 이 숫자 코리안넷에 검색한다. 검색 작업은 데이터 품질에 따라 시간이 얼마나 소요될 지 모르기 때문에 비동기 백그라운드 작업으로 수행한다. 이를 위해 NetworkTask를 생성해서 실행시킨다.

@Override  
protected BarcodeData doInBackground(Void... params) {  
 String name = null; // 요청 결과를 저장할 변수.  
 String imgSrc = null;  
 try {  
 Document doc = null;  
 //전체 html문서 획득  
 doc = Jsoup.*connect*("http://www.koreannet.or.kr/home/hpisSrchGtin.gs1?gtin=" + number).get();  
 //파싱  
 Elements title = doc.select(".productTit"); //class명이 productTit인 것을 선택  
 if(!title.text().isEmpty()) {  
 name = title.text().substring(14); //13자리 GTIN번호와 공백 하나 제거  
 Log.*d*("doInBackground", "name:" + name);  
 }  
 imgSrc = doc.select("#detailImage").attr("src"); //id가 detailImage 것에서 src속성 선택  
 if(imgSrc.contains("no\_img")) //대표사진 없는 물품인 경우  
 imgSrc = null;  
 //barcodeData.setImageSrc(imgSrc);  
 Log.*d*("doInBackground", "imgSrc:" + imgSrc);  
  
 if(imgSrc != null) {  
 URL url = new URL(imgSrc);  
 URLConnection conn = url.openConnection();  
 conn.connect();  
 BufferedInputStream bis = new BufferedInputStream(conn.getInputStream());  
 thumbImage = BitmapFactory.*decodeStream*(bis);  
 isImageChanged = true;  
 bis.close();  
 }  
 } catch (MalformedURLException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return new BarcodeData(-1, number, name, imgSrc, null, null); //아직 저장버튼 누른건 아니므로 itemNum = -1  
}

NetworkTask의 백그라운드 메소드는 코리안넷에 바코드를 검색한 결과를 html로 얻은 다음 그 안에서 이름과 대표사진을 파싱하는 동작을 한다. http통신과 파싱은 Jsoup라이브러리를 이용했다.

@Override  
protected void onPostExecute(BarcodeData data) {  
 super.onPostExecute(data);  
 EditText editText = findViewById(R.id.*editText\_name*);  
 TextView noticeText = findViewById(R.id.*textView\_Notice*);  
 if(data.getName() != null) {  
 noticeText.setVisibility(View.*GONE*);  
 editText.setText(data.getName());  
 barcodeData.setName(data.getName()); //검색된 품목명으로 초기화  
 //대표 이미지를 썸네일로 설정  
 if(data.getImageSrc() != null) {  
 ImageView imageView = findViewById(R.id.*imageView\_setThumb*);  
 imageView.setImageBitmap(thumbImage);  
 imageView.setScaleType(ImageView.ScaleType.*CENTER\_CROP*);  
 }  
 } else {  
 noticeText.setText("검색된 상품이 없습니다!\n일반 촬영 모드를 선택해주세요");  
 noticeText.setVisibility(View.*VISIBLE*);  
 }  
 progressDialog.dismiss(); //검색중 다이얼로그 해제  
}

백그라운드에서 검색이 완료되면 onPostExecute를 호출한다. 검색 결과로부터 사진과 이름을 SetScanInfoActivity의 뷰에 반영하는 동작을 한다.

//dday설정 달력, 일수 클릭 이벤트  
public void setDateBtnOnClick(View view) {  
 DialogFragment newFragment = new SelDateFragment();  
 newFragment.show(getSupportFragmentManager(), "dateSel");  
}  
public void processDatePickerResult(int year, int month, int day){  
 //달력에서 선택한 날짜로 dday를 구해서 출력  
 String month\_string = Integer.*toString*(month+1);  
 String day\_string = Integer.*toString*(day);  
 String year\_string = Integer.*toString*(year);  
 barcodeData.setDday(year\_string + "-" + month\_string + "-" + day\_string);  
 //textview에는 dday 수치를 보여줘야함  
 TextView textDday = findViewById(R.id.*textView\_dday*);  
 textDday.setText(getDday(year, month, day) + "일 남았습니다");  
}

재료의 dday를 설정하기 위해 달력 아이콘을 클릭하면 setDateBtnOnClick()가 호출된다. setDateBtnOnClick()는 달력 다이얼로그를 상속받은 SelDateFragment를 보여주고 사용자가 달력에서 원하는 날짜를 선택하면 processDatePickerResult의 매개변수에 년, 월, 일이 들어간다. 이 날짜 정보를 이용해 dday계산을 한다.

//완성된 BarcodeData를 MainActivity에 전달  
public void setSavebtnOnclick(View v) {  
 if(barcodeData.getName() == null || barcodeData.getName().isEmpty()) {  
 Toast.*makeText*(getApplicationContext(), "이름을 입력해주세요", Toast.*LENGTH\_LONG*).show();  
 return;  
 } else if(barcodeData.getDday() == null) {  
 Toast.*makeText*(getApplicationContext(), "기간을 설정해주세요", Toast.*LENGTH\_LONG*).show();  
 return;  
 }//return condition  
  
 //make intent  
 Intent intent = new Intent(SetScanInfoActivity.this, MainActivity.class);  
 if(mode.equals("add")) {  
 barcodeData.setRegDate(*nowTimetoString*());  
 barcodeData.setItemNum(System.*currentTimeMillis*()); //생성될 아이템의 기본키 값 시간으로 사용  
 } else if(mode.equals("edit")) {  
 intent.putExtra("index", this.list\_index);  
 }  
 //비트맵 유효하면 thumbImage를 파일로 저장하고 경로 획득  
 if(thumbImage != null) {  
 if(isImageChanged) { //바코드로 사진 얻었거나 일반 촬영으로 사진 얻은 경우  
 barcodeData.setImageSrc(saveBitmap(thumbImage, Long.*toString*(barcodeData.getItemNum())));  
 }  
 }  
 intent.putExtra("barcodedata", barcodeData);  
 setResult(1, intent);  
 finish();  
}

사용자가 저장 버튼을 누르면 setSavebtnOnclick()이 호출된다. 이전의 바코드 스캔 또는 사진촬영, dday설정이 성공적으로 완료되었다면 SetScanInfoActivity의 멤버변수인 barcodeData에 정보들이 저장되어있다. barcodeData의 itemNum에는 유일함을 식별하기 위해 현재 시간의 ms(밀리초)를 얻어와 기본키 값으로 사용한다. 한 명의 사용자가 한 개의 ScanRemind앱을 실행하기 때문에 중복이 없다고 볼 수 있다. 미리보기 이미지도 유효한 것이 확인되면 내부 저장소(Cache)에 저장함과 동시에 메인 액티비티로 barcodeData를 인탠트에 넣어보낸다

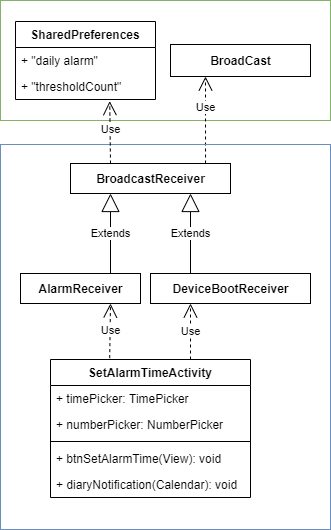


그림 ４. SetAlarmTime 액티비티

SetAlarmTimeActivity가 알림 시간을 설정하고 해당 시간에 노티피케이션 해주는 동작을 소스코드와 함께 설명한다.

//앞서 설정한 값을 사용. 없으면 디폴트 값은 현재시간  
SharedPreferences sharedPreferences = getSharedPreferences("daily alarm", *MODE\_PRIVATE*);  
long millis = sharedPreferences.getLong("nextNotifyTime", Calendar.*getInstance*().getTimeInMillis());  
int ddaynum = sharedPreferences.getInt("ddayThreshold", 1);

//Preference에 설정한 값 저장  
SharedPreferences.Editor editor = getSharedPreferences("daily alarm", *MODE\_PRIVATE*).edit();  
editor.putLong("nextNotifyTime", (long)calendar.getTimeInMillis());  
editor.putInt("ddayThreshold", numberPicker.getValue());  
editor.apply();

SetAlarmTimeActivity()는 공유변수(SharedPreferences)에 저장된 값을 사용한다. 공유변수는 key-value 형태로 저장되어 있는데 알림 시간을 의미하는 "nextNotifyTime"과 얼만큼 남은 dday를 표시할지 기준치가 되는 "ddayThreshold"를 저장한다.

@Override  
public void onReceive(Context context, Intent intent) {  
 // Preference에서 임박 항목 갯수 가져오기  
 SharedPreferences sharedPreferences = context.getSharedPreferences("daily alarm", *MODE\_PRIVATE*);  
 int thresholdCount = sharedPreferences.getInt("thresholdCount", -1);  
  
 builder.setAutoCancel(true)  
 .setDefaults(NotificationCompat.*DEFAULT\_ALL*)  
 .setWhen(System.*currentTimeMillis*())  
 .setTicker("{Time to watch some cool stuff!}")  
 .setContentTitle("냉장고를 확인해주세요!")  
 .setContentText("기간이 임박한 재료가 " + Integer.*toString*(thresholdCount) + "개 있습니다.")  
 .setContentInfo("INFO")  
 .setContentIntent(pendingI);  
  
 if (notificationManager != null) {  
 // 1개 이상이면 노티피케이션 동작시킴  
 if(thresholdCount > 0)  
 notificationManager.notify(1234, builder.build());  
  
 Calendar nextNotifyTime = Calendar.*getInstance*();  
  
 //내일 같은 시간으로 알람시간 결정  
 nextNotifyTime.add(Calendar.*DATE*, 1);  
  
 //Preference에 설정한 값 저장  
 SharedPreferences.Editor editor = context.getSharedPreferences("daily alarm", *MODE\_PRIVATE*).edit();  
 editor.putLong("nextNotifyTime", nextNotifyTime.getTimeInMillis());  
 editor.apply();  
 }

위 코드는 AlarmReceiver의 onReceive() 이벤트 메소드 중 일부분이다. AlarmReceiver가 지정했던 시간이 되어 전파된 알람 broadcast를 수신하면 onReceive() 메소드가 호출된다. 노티피케이션으로 기간한 임박한 재료 개수를 사용자에게 알려준다. 그리고 알람을 매일 동작시키기 위해서 nextNotifyTime을 현재 시작으로 다시 설정함으로써 매일 같은 시작에 알람 broadcast가 전파된다.

마치며 (결론 및 느낀점 적는 부분)

일상에서 낭비되는 식재료를 관리하는 리마인더 앱을 제작했다. 바코드를 이용해서 식재료를 쉽게 등록할 수 있지만, 검색되지 않는 바코드도 있고 아예 바코드가 없는(과일 및 야채)들을 발견했다. 따라서 이런 품목들도 관리할 수 있게끔 일반 촬영 모드를 추가했다.

이렇게 해서 완성된 어플을 사용해보니 식재료에 한정되지 않고 바코드가 있거나, 없거나 품목을 등록해서 사용할 수 있다는 점을 알 수 있었다. 즉, 사용 범위가 냉장고-식재료 라는 특수한 범위에서 일상-모든 품목 범위로 일반적인(General) 경우들을 다룰 수 있게 되었다.

일반적인 경우에서 사용자 경험을 발전시킬 수 있는 아이디어로 노티피케이션뿐만 아니라 홈 화면에 위젯을 추가해서 상시로 확인 할 수 있게 하면 리마인더앱 방향성 유지에 좋을 것 같다. 또는 그룹 기능으로 ‘냉장고’ 그룹의 리스트, ‘김치냉장고’ 그룹의 리스트, ‘옷장’ 그룹의 리스트를 만들어서 사용할 수 있다면 여러 범위를 커버할 수 있을것이다.