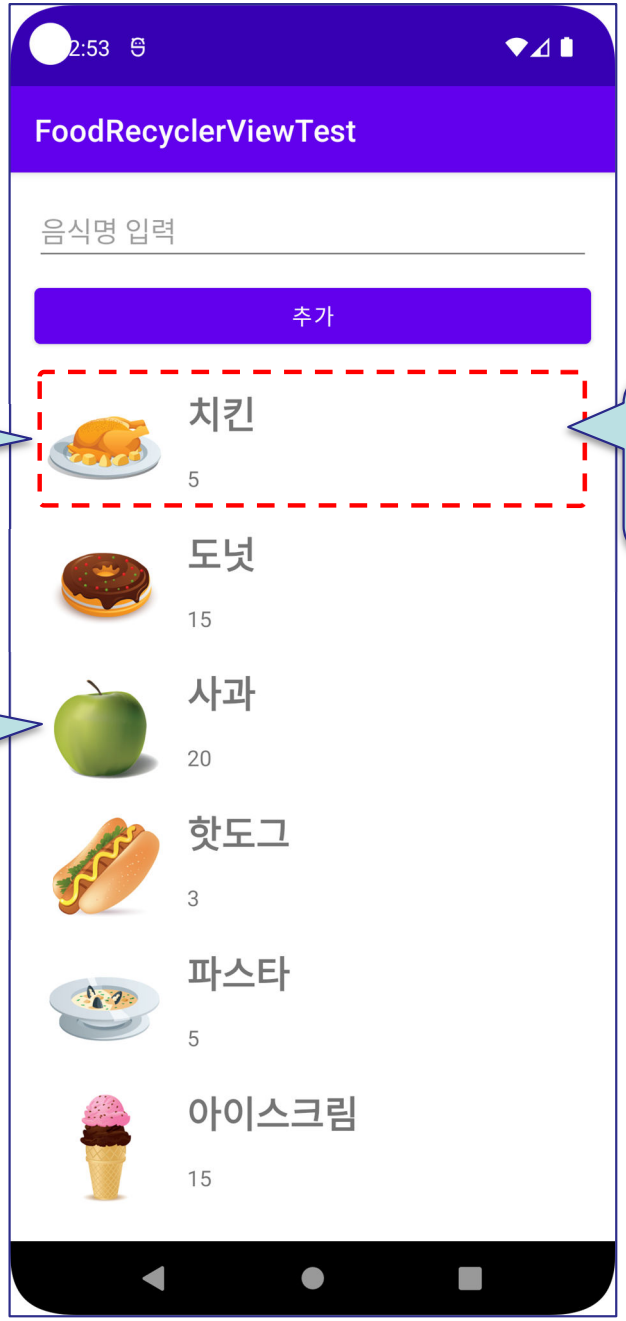
RecyclerView 02

- RecyclerView의 확장
- 커스텀 이벤트 리스너의 구현

- http://developer.android.com

확장한 RecyclerView의 예

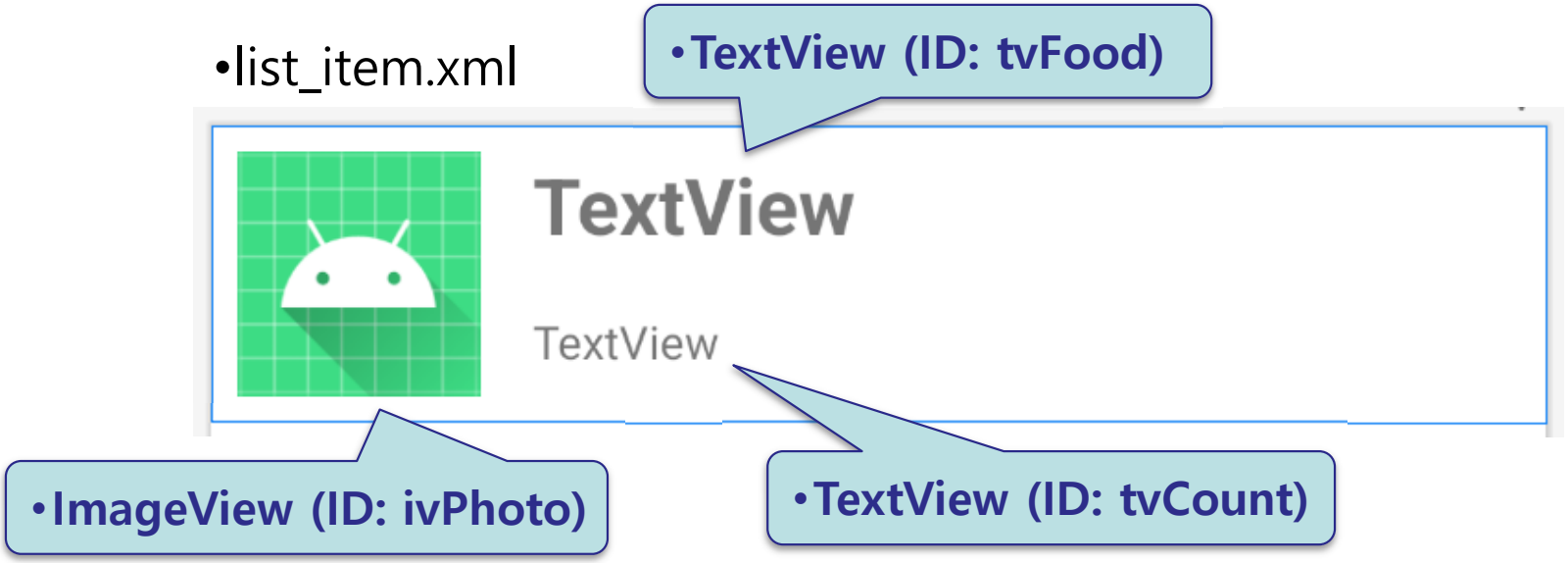


- list_item.xml
- ImageView
- TextView
- TextView
- 이미지의 경우 res 의 drawable 에 미리 저장한 이미지 파일 사용
- Item 항목 하나를 표현하기 위해 3개의 데이터 값 필요

1. Item View의 레이아웃 작성

- RecyclerView 의 Item을 표현할 레이아웃 작성

- res/layout/*list_item.xml* 추가
  - ConstraintLayout 사용
    - layout_width: match_parent, layout_height: wrap_content 지정



2. DTO 의 작성 및 샘플 데이터 추가

- Item 항목 하나는 이미지와 두 개의 텍스트의 집합 → 항목 하나당 photo, food, count 필요

- DTO(Data Transfer Object) 구현 (클래스 추가)
  - 관련 있는 여러 데이터 값을 하나로 묶어 다른 계층에 전송하는 용도로 사용하는 Design Pattern
  - *data class* 사용 → getter/setter 등 자동 추가

•FoodDto

•list_item view 에 표시할 데이터 항목

•count 만 변경 가능한 상태

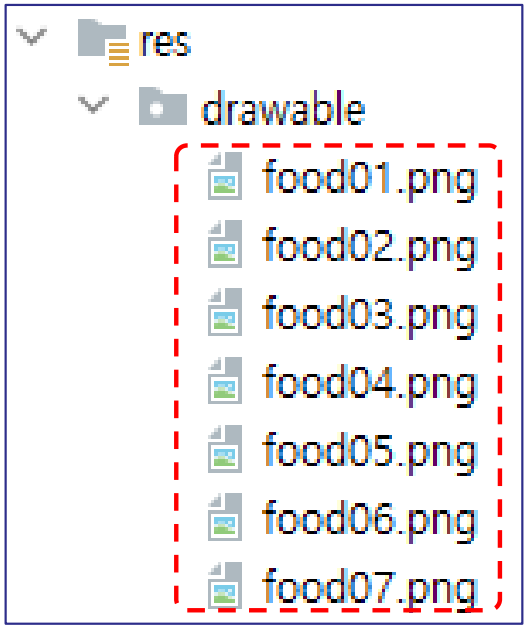
```
3   data class FoodDto (val photo : Int, val food : String, var count: Int) {
4   //    override fun toString(): String {
5   //        return "$food ($count)"
6   //    }
7       override fun toString() = "$food ($count)"
8   }
```

•리소스 ID는 정수형

•toString() 재정의 : kotlin 문법 적용

2. DTO 의 작성 및 샘플 데이터 추가

샘플 이미지 추가



• 이미지 파일 추가

FoodDao 구현 (클래스 추가)

•FoodDao

```
class FoodDao {
    val foods = ArrayList<FoodDto>()

    init {
        foods.add (FoodDto(R.drawable.food01, "치즈", 10))
        foods.add (FoodDto(R.drawable.food02, "치킨", 5))
        foods.add (FoodDto(R.drawable.food03, "도넛", 15))
        foods.add (FoodDto(R.drawable.food04, "사과", 20))
        foods.add (FoodDto(R.drawable.food05, "핫도그", 3))
        foods.add (FoodDto(R.drawable.food06, "파스타", 5))
        foods.add (FoodDto(R.drawable.food07, "아이스크림", 15))
    }
}
```

• 리소스 ID 는 정수형

FoodDao 의 사용

```
val dao = FoodDao()
val foods = dao.foods
```

3. Adapter 구현

Adapter 클래스 추가 후 ViewHolder 구현

- Adapter 내부 클래스로 생성

•FoodAdapter

• Adapter 에 사용할 데이터

• RecyclerView.Adapter<T> 상속

```
class FoodAdapter (val foods : ArrayList<String>)
    : RecyclerView.Adapter<FoodAdapter.FoodViewHolder>() {

    ...

    class FoodViewHolder(view: View) : RecyclerView.ViewHolder(view) {
        val photo = view.findViewById<ImageView>(R.id.ivPhoto)
        val food = view.findViewById<TextView>(R.id.tvFood)
        val count = view.findViewById<TextView>(R.id.tvCount)
    }
}
```

• list_item.xml 로 생성한 view 객체

• RecyclerView.ViewHolder 상속

• list_item 레이아웃을 구성하는 각 view 를 찾아 멤버변수로 보관

3. Adapter 구현

Adapter 의 멤버함수 재정의

•FoodAdapter

```
class FoodAdapter (val foods : ArrayList<FoodDto>)
    : RecyclerView.Adapter<FoodAdapter.FoodViewHolder>() {

    /*재정의 필수 - 데이터의 개수 확인이 필요할 때 호출*/
    override fun getItemCount(): Int = foods.size
    /*재정의 필수 - 각 item view 의 view holder 생성 시 호출*/
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): FoodViewHolder {
        val itemView = LayoutInflater.from(parent.context).inflate(R.layout.list_item, parent, false)
        return FoodViewHolder(itemView)
    }
    /*재정의 필수 - 각 item view 의 항목에 데이터 결합 시 호출*/
    override fun onBindViewHolder(holder: FoodViewHolder, position: Int) {
        holder.photo.setImageResource( foods[position].photo )
        holder.food.text = foods[position].food
        holder.count.text = foods[position].count.toString()
    }

    class FoodViewHolder(view: View) : RecyclerView.ViewHolder(view) {...}
}
```

•context 사용

•list_item 레이아웃 지정

•ViewHolder 의 멤버변수로 지정한 view 에 데이터 결합

•FoodDto의 count 는 정수형이므로 문자열로 변환

4. RecyclerView 설정

RecylcerView 와 Adapter 의 연결

•MainActivity

```
override fun onCreate(savedInstanceState: Bundle?) {
    ...
    /*데이터 준비*/
    val foods = FoodDao().foods

    /*어댑터 생성*/
    val adapter = FoodAdapter(foods)

    /*레이아웃매니저 생성 및 설정*/
    val layoutManager = LinearLayoutManager(this)
    layoutManager.orientation = LinearLayoutManager.VERTICAL

    /*RecyclerView 에 레이아웃매니저 및 Adapter 설정*/
    binding.recyclerView.layoutManager = layoutManager
    binding.recyclerView.adapter = adapter
}
```

Binding 의 적용

findViewById() 대신 Binding 사용

```
class BindingFoodAdapter (val foods : ArrayList<FoodDto>)
    : RecyclerView.Adapter<BindingFoodAdapter.FoodViewHolder>() {

    /*재정의 필수 - 데이터의 개수 확인이 필요할 때 호출*/
    override fun getItemCount(): Int = foods.size

    /*재정의 필수 - 각 item view 의 view holder 생성 시 호출*/
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): FoodViewHolder {
        val itemBinding = ListItemBinding.inflate(LayoutInflater.from(parent.context), parent, false)
        return FoodViewHolder(itemBinding)
    }

    /*재정의 필수 - 각 item view 의 항목에 데이터 결합 시 호출*/
    override fun onBindViewHolder(holder: FoodViewHolder, position: Int) {
        holder.itemBinding.ivPhoto.setImageResource( foods[position].photo )
        holder.itemBinding.tvFood.text = foods[position].food
        holder.itemBinding.tvCount.text = foods[position].count.toString()
    }

    class FoodViewHolden(val itemBinding: ListItemBinding) : RecyclerView.ViewHolden(itemBinding.root)

}
```

- BindingFoodAdapter 추가
- 사용 시 MainActivity 에서 사용어댑터 변경

- list_item.xml 을 위해 생성된 ListItemBinding 클래스

- Binding 사용 방식으로 변경

- ListItemBinding 객체를 멤버변수로 지정

- ListItemBinding 객체의 root 객체를 전달

실습

- 다음과 같은 레이아웃을 갖도록 Adapter 를 구현
  - 기존 프로젝트에 아래의 항목 추가 및 구현 후 MainActivity 에서 기존 정보 교체
  - 레이아웃: list_weather.xml

  - Adapter 명: CustomAdapter
  - DTO 명 : WeatherDto
  - DAO 명 : WeatherDao

이벤트 핸들러의 구현 1

기본 방법

- ViewHolder 의 생성자에서 각 View 에 이벤트 리스너 추가

Model View Controller

- 이벤트 처리에 필요한 변수들을 생성자의 매개변수로 전달

```
class FoodViewHolder(val itemBinding: ListItemBinding, foods: ArrayList<FoodDto>)
        : RecyclerView.ViewHolder(itemBinding.root) {
    val TAG = "FoodViewHolder"
    init {
        itemBinding.root.setOnClickListener { it: View!
            Log.d(TAG, "${foods[adapterPosition]}")
        }
    }
}
```

- List Item View를 구성하는 각 View에 이벤트 리스너 설정

이벤트 핸들러의 구현 2

커스텀 리스너의 구현

- 이벤트 핸들러의 구현 부분을 외부에서 생성하여 이벤트가 발생하는 부분에 연결
- 리스너가 정의되어 있을 경우 (e.g. Button)

```
val listener = object : View.OnClickListener {
    override fun onClick(view: View?) {
        // 수행할 기능을 구현
    }
}

/*button 클릭 시 호출되는 함수에서 listener의 onClick 부분을 실행*/
binding.button.setOnClickListener( listener )
```

- 버튼 클릭 시 수행해야 할 부분을 버튼 외부에 Listener 로 구현

- 버튼에 생성한 Listener 전달

- RecyclerView.Adapter 는 리스너가 정의되어 있지 않으므로 직접 해당부분을 구현해야 함

이벤트 핸들러의 구현 3

Adapter 추가 항목 – itemClick 이라고 가정할 경우

- 이벤트 Listener Interface : 구현할 Listener 정의 제공용
- 멤버변수 – 외부에서 전달받은 listener 를 보관할 멤버 변수
- set 멤버함수 – listener 를 전달받는 멤버함수

• Listener 를 지정하는 인터페이스 선언

• 필요에 따라 매개변수 추가

```
interface OnItemClickListener {
    fun onItemClick(view : View, position: Int) : Unit
}

lateinit var listener: OnItemClickListener

fun setOnItemClickListener (listener: OnItemClickListener) {
    this.listener = listener
}
```

• 전달받은 Listener 를 보관할 멤버변수

• 외부에서 구현한 listener 를 전달받아 멤버변수에 저장

이벤트 핸들러의 구현 4

ViewHolder 구현 항목

- 이벤트 처리 listener 에 외부에서 전달한 listener 의 멤버함수 호출

```
class FoodViewHolder(val itemBinding: ListItemBinding, listener: OnItemClickListener )
        : RecyclerView.ViewHolder(itemBinding.root) {
    init {
        itemBinding.root.setOnClickListener ( object: View.OnClickListener {
            override fun onClick(view: View?) {
                listener.onItemClick(view!!, adapterPosition)
            }
        })
    }
}
```

- 생성자의 매개변수로 외부에서 생성한 Listener 전달
- 실제 이벤트 발생 시 처리되는 이벤트 Listener
- 전달받은 listener 의 멤버함수 호출

```
class FoodViewHolder(val itemBinding: ListItemBinding, listener: OnItemClickListener )
        : RecyclerView.ViewHolder(itemBinding.root) {
    init {
        itemBinding.root.setOnClickListener { it: View!
            listener.onItemClick(it, adapterPosition)
        }
    }
}
```

- 람다함수로 표현

이벤트 핸들러의 구현 5

외부에서 Listener 생성 후 전달

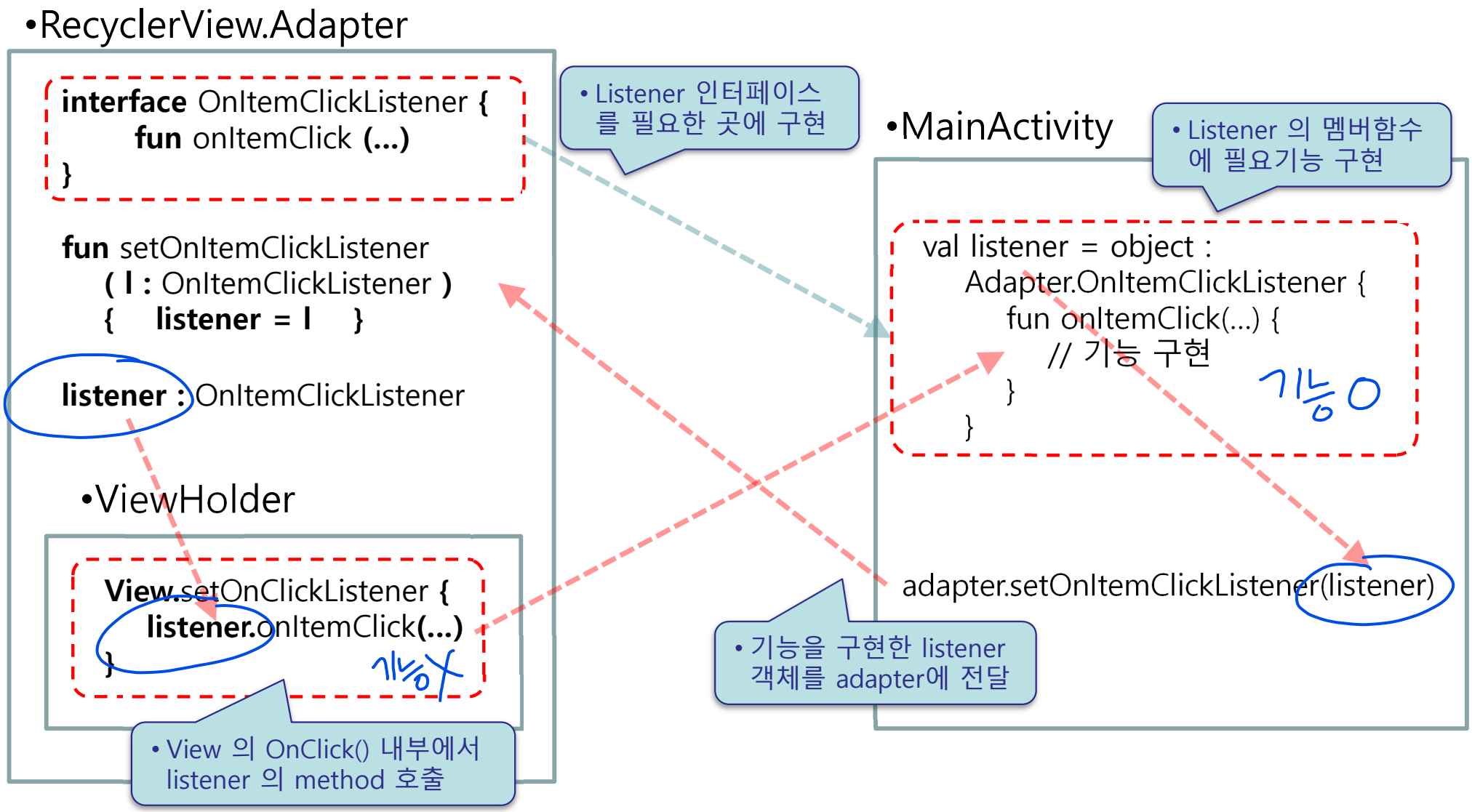
•MainActivity

```
val foods = FoodDao().foods     // 데이터 준비

val adapter = BindingFoodAdapter(foods)     // 어댑터 생성

val TAG = "MainActivity"
/*커스텀 리스너 사용*/
adapter.setOnItemClickListener(object : BindingFoodAdapter.OnItemClickListener {
    override fun onItemClick(view: View, position: Int) {
        Log.d(TAG, "${foods[position]}")
    }
})
```

• 커스텀 리스너 객체 생성 후 Adapter 에 설정

이벤트 핸들러의 구현 3

커스텀 리스너의 구현 개념



실습

- Adapter의 항목을 롱클릭 할 경우 삭제를 수행하는 이벤트 리스너를 커스텀 리스너로 구현
  - FoodAdapter 로 교체 후 진행
  - 이벤트 명 : ItemLongClick

- 추가 내용
  - 삭제 시 삭제확인을 요청하는 대화상자(AlertDialog)를 표시하고 [확인] 일 경우에만 삭제하도록 기능 추가