Android MVI architecture with jetpack & coroutines/Flow

Creating a base adapter item

By doing this we will be able to have a single interface to implement and use as common configuration for our adapter's items.

```
interface ViewBindingAdapterItem {
    val itemViewType: Int
}
```

This item only includes an itemViewType property so that we can have polymorphic RecyclerView adapters. We only need this property for now, but we can surely add more in order to enrich our base functionality in the future.

Creating a base ViewHolder

Our base ViewHolder will be extended from each ViewHolder in each of our adapters.

```
import androidx.recyclerview.widget.RecyclerView
import androidx.viewbinding.ViewBinding

abstract class ViewBindingViewHolder<Item : ViewBindingAdapterItem, out VB : ViewBinding>(
    protected val binding: VB
) : RecyclerView.ViewHolder(binding.root) {
    abstract fun bind(item: Item)
    open fun bind(item: Item, payloads: List<Any>) {
        if (payloads.isEmpty()) {
            bind(item = item)
        }
    }
}
```

Creating a base adapter

Our base adapter will extend from `ListAdapter` since we want to use `DiffUtil` for diffing in our RecyclerViews

```
import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.recyclerview.widget.ListAdapter
import androidx.viewbinding.ViewBinding

abstract class ViewBindingAdapter<Item : ViewBindingAdapterItem, VB : ViewBinding>(
    diffCallback: ViewBindingDiffUtilCallback<Item>
) : ListAdapter<Item, ViewBindingViewHolder<Item, VB>>(diffCallback) {
    override fun onBindViewHolder(holder: ViewBindingViewHolder<Item, VB>, position: Int) =
        holder.bind(item = getItem(position))

    override fun onBindViewHolder(
        holder: ViewBindingViewHolder<Item, VB>,
        position: Int,
        payloads: MutableList<Any>
    ) {
        holder.bind(item = getItem(position), payloads = payloads)
    }

    override fun getItemViewType(position: Int): Int = getItem(position).itemViewType

    protected val ViewGroup.layoutInflater: LayoutInflater
        get() = LayoutInflater.from(this.context)
}
```

The implementation we have here just delegates the `onBindViewHolder` methods to our own ViewHolders.
We also provide an extension function for easier access to the layout inflater when creating a new adapter.

Creating a base DiffUtil callback

Our base DiffUtil callback will extend from `DiffUtil.ItemCallback` and use `ViewBindingAdapterItem` as a generic input.

```
abstract class ViewBindingDiffUtilCallback<Item : ViewBindingAdapterItem> :
    DiffUtil.ItemCallback<Item>()
```