Programación multimedia y dispositivos móviles

Actividad 3.2. Patrón MVVM

Francisco José García Cutillas | 2FPGS_DAM

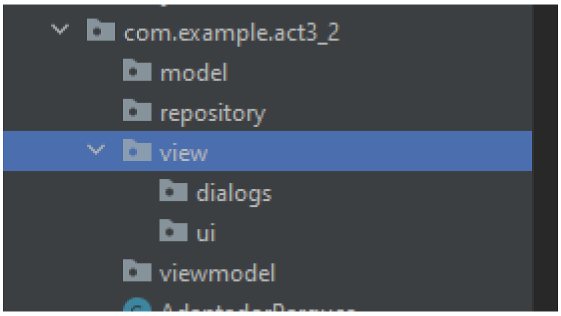
Índice

Ejercicio 1

Ahora vamos a aplicar el patrón MVVM a nuestro proyecto. Para ello, crea un ViewModel y traslada las diferentes funcionalidades de la aplicación a su capa correspondiente.

En este caso trataremos únicamente el primer fragmento, que contiene el RecyclerView, y dejaremos el segundo fragmento para actividades posteriores.

Para comenzar con a la adaptación al patrón MVVM vamos a generar la siguiente estructura de ficheros:



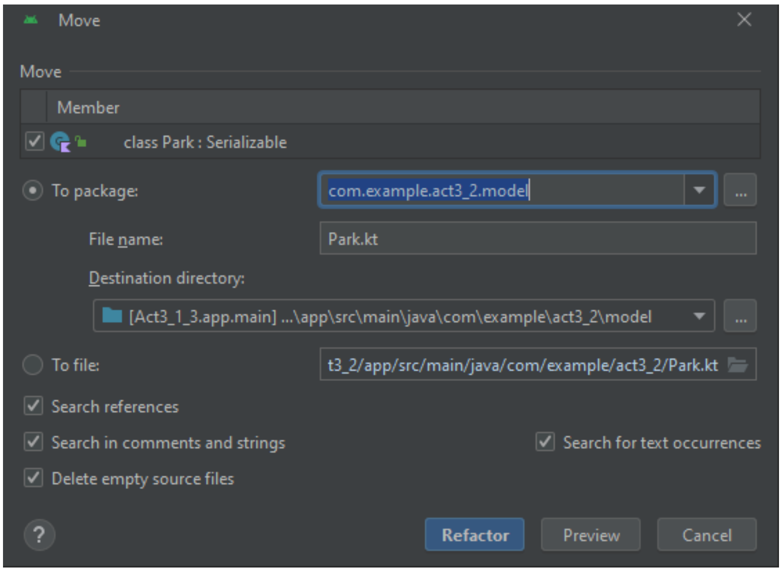
```
com.example.act3_2
    model
    repository
    view
        dialogs
        ui
    viewmodel
```

Con esto hemos generado:

- Paquete model para almacenar los modelos de datos.
- Paquete repository para ofrecer un único punto de acceso a los datos.
- Paquete view que es donde se guardarán las clases relacionadas con las vistas. Que a su vez contiene:
    - Paquete ui, encargado de la interfaz.
    - Paquete dialogs, encargado de la gestión de los diálogos.
- Paquete viewmodel que es donde se gestiona toda la lógica de la vista.

Comenzamos con el paquete model.

A él vamos a arrastrar la clase “Park” y el objeto “Parks”.



☑ class Park : Serializable

● To package: com.example.act3_2.model

File name: Park.kt

Destination directory: [Act3_1_3.app.main] ...\app\src\main\java\com\example\act3_2\model

○ To file: t3_2/app/src/main/java/com/example/act3_2/Park.kt

☑ Search references

☑ Search in comments and strings

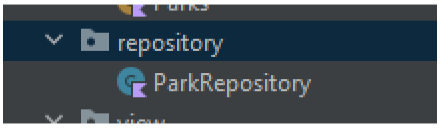
☑ Search for text occurrences

☑ Delete empty source files

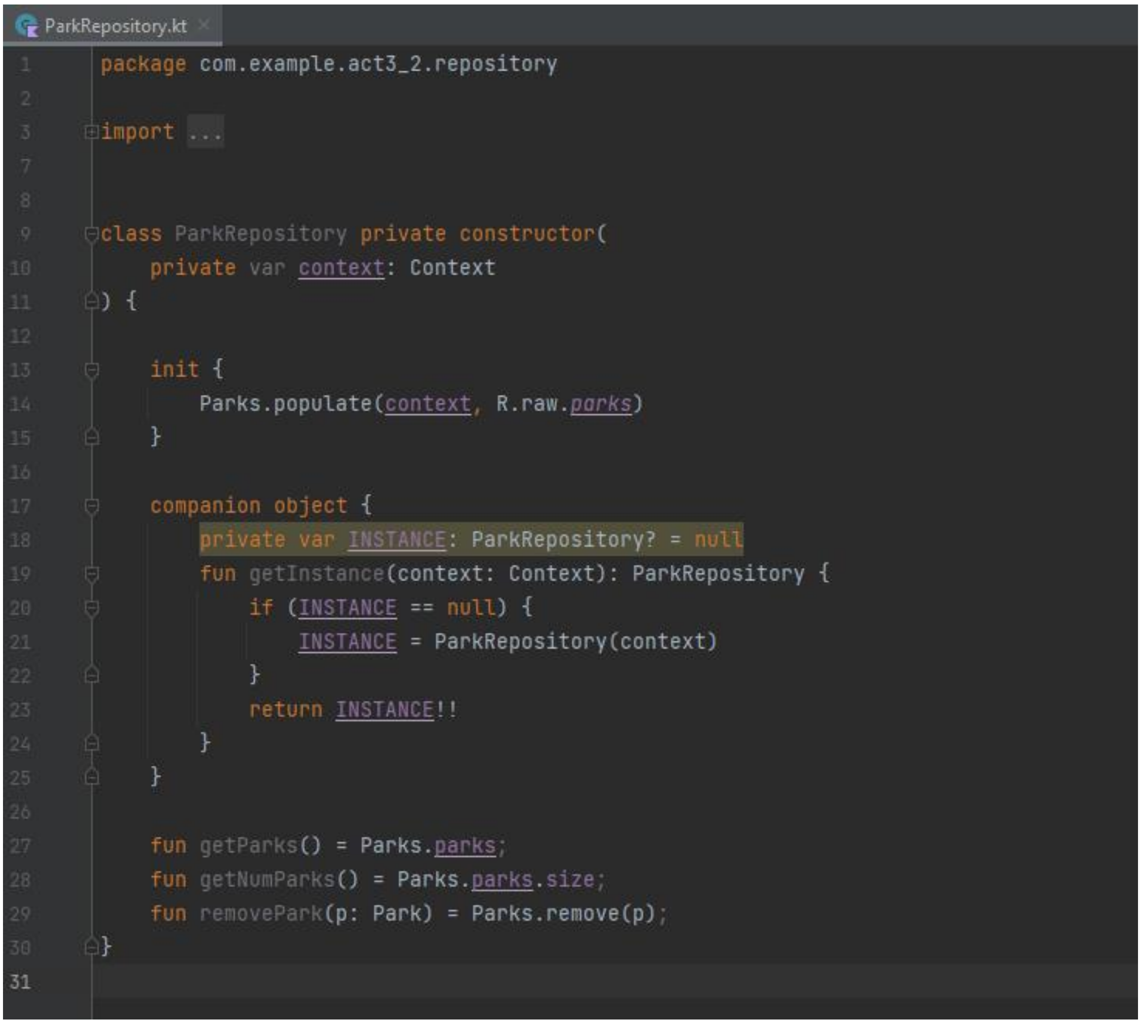
En ambos casos Android Studio nos sugiere que hay que refactorizar el código del paquete en el que está contenida la clase y el objeto. Le damos a "refactor" en ambos casos.

Paquete repository

Dentro de este paquete vamos a crear una Kotlin class "ParkRepository", que va a ser la encargada de gestionar el acceso al modelo.



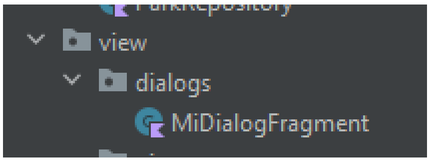
El código de esta clase será el siguiente:



```kotlin
package com.example.act3_2.repository

import ...


class ParkRepository private constructor(
    private var context: Context
) {

    init {
        Parks.populate(context, R.raw.parks)
    }

    companion object {
        private var INSTANCE: ParkRepository? = null
        fun getInstance(context: Context): ParkRepository {
            if (INSTANCE == null) {
                INSTANCE = ParkRepository(context)
            }
            return INSTANCE!!
        }
    }

    fun getParks() = Parks.parks;
    fun getNumParks() = Parks.parks.size;
    fun removePark(p: Park) = Parks.remove(p);
}

```

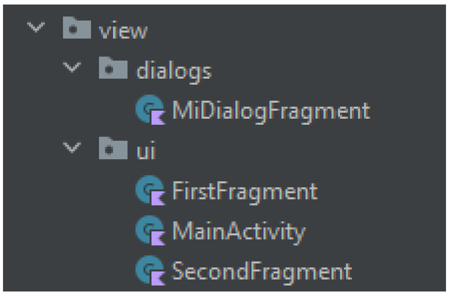
Paquete view

Dentro del fichero “dialogs” vamos a mover la clase “MiDialogFragment”, que al igual que en el caso anterior, debemos refactorizar el paquete en el que se va a contener ésta.



```
view
  dialogs
    MiDialogFragment
```

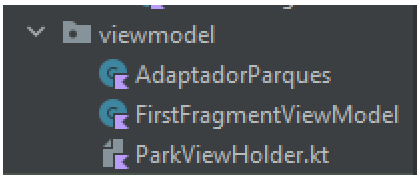
Con respecto al fichero “ui”, en él vamos a meter todo lo relacionado con la interfaz. Que son las clases “FirstFragment”, “SecondFragment” y “MainActivity”, que al igual que el caso anterior debemos refactorizar la ubicación de la clase, para quedar finalmente una estructura como la siguiente:



```
view
  dialogs
    MiDialogFragment
  ui
    FirstFragment
    MainActivity
    SecondFragment
```

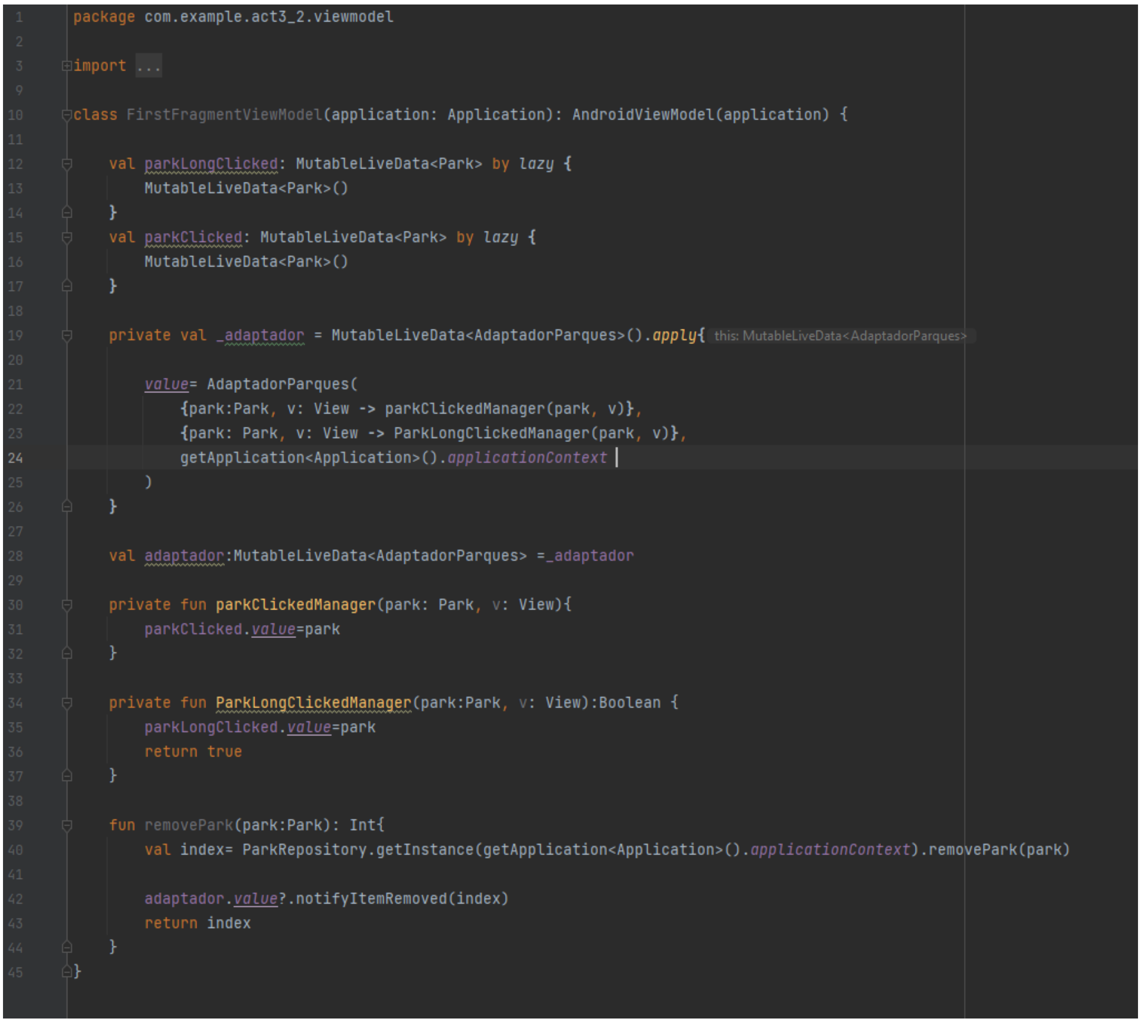
Paquete viewmodel

En él vamos a meter las clases que nos quedan, “AdaptadorParques” y “ParkViewHolder.kt”. También vamos a crear una clase “FirstFragmentViewModel”, que será lo equivalente al ViewModel para el primer fragmento.



```
viewmodel
  AdaptadorParques
  FirstFragmentViewModel
  ParkViewHolder.kt
```

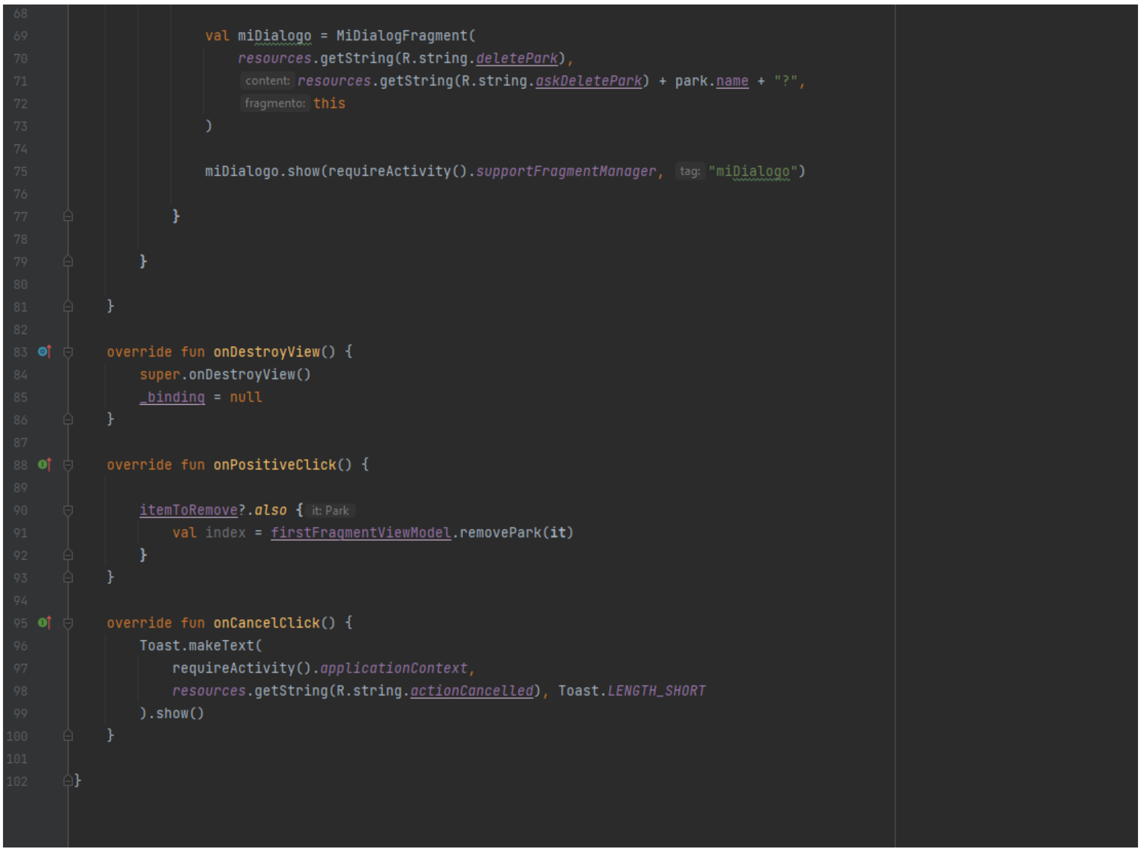
El código de la clase “FirstFragmentViewModel” quedaría de la siguiente forma:



```kotlin
package com.example.act3_2.viewmodel

import ...

class FirstFragmentViewModel(application: Application): AndroidViewModel(application) {

    val parkLongClicked: MutableLiveData<Park> by lazy {
        MutableLiveData<Park>()
    }
    val parkClicked: MutableLiveData<Park> by lazy {
        MutableLiveData<Park>()
    }

    private val _adaptador = MutableLiveData<AdaptadorParques>().apply{

        value= AdaptadorParques(
            {park:Park, v: View -> parkClickedManager(park, v)},
            {park: Park, v: View -> ParkLongClickedManager(park, v)},
            getApplication<Application>().applicationContext
        )
    }

    val adaptador:MutableLiveData<AdaptadorParques> =_adaptador

    private fun parkClickedManager(park: Park, v: View){
        parkClicked.value=park
    }

    private fun ParkLongClickedManager(park:Park, v: View):Boolean {
        parkLongClicked.value=park
        return true
    }

    fun removePark(park:Park): Int{
        val index= ParkRepository.getInstance(getApplication<Application>().applicationContext).removePark(park)

        adaptador.value?.notifyItemRemoved(index)
        return index
    }
}
```

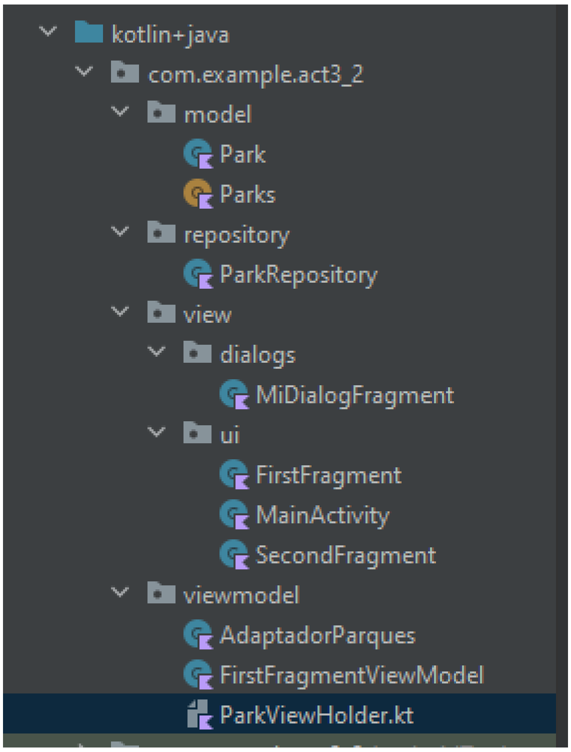
Tras realizar esta clase, ahora debemos realizar también unas modificaciones en la clase “FirstFragment” del paquete view -> ui, encargada de gestionar la interfaz del primer fragmento. Esta clase quedaría de la siguiente manera:

```kotlin
package com.example.act3_2.view.ui
import ...

class FirstFragment : Fragment(), MiDialogFragment.OnOKOrCancelListener {

    private var itemToRemove: Park?

    init {
        itemToRemove = null
    }

    private lateinit var firstFragmentViewModel: FirstFragmentViewModel

    private var _binding: FragmentFirstBinding? = null
    private val binding get() = _binding!!

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {

        _binding = FragmentFirstBinding.inflate(inflater, container, false)

        return binding.root

    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        firstFragmentViewModel =
            ViewModelProvider(this).get(FirstFragmentViewModel::class.java)

        binding.ParksRecyclerView.layoutManager = LinearLayoutManager(requireActivity())

        firstFragmentViewModel.adaptador.observe(viewLifecycleOwner) {
            binding.ParksRecyclerView.adapter = it
        }

        firstFragmentViewModel.parkClicked.observe(viewLifecycleOwner) { park ->
            park?.let {
                firstFragmentViewModel.parkClicked.value = null

                val bundle = bundleOf("park" to park)
                findNavController().navigate(R.id.action_FirstFragment_to_SecondFragment, bundle)
            }
        }

        firstFragmentViewModel.parkLongClicked.observe(viewLifecycleOwner) { park ->
            park?.let {
                firstFragmentViewModel.parkLongClicked.value = null

                itemToRemove = park
```



```kotlin

                val miDialogo = MiDialogFragment(
                    resources.getString(R.string.deletePark),
                    content: resources.getString(R.string.askDeletePark) + park.name + "?",
                    fragmento: this
                )

                miDialogo.show(requireActivity().supportFragmentManager,  tag: "miDialogo")

            }

        }

    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }

    override fun onPositiveClick() {

        itemToRemove?.also { it: Park
            val index = firstFragmentViewModel.removePark(it)
        }
    }

    override fun onCancelClick() {
        Toast.makeText(
            requireActivity().applicationContext,
            resources.getString(R.string.actionCancelled), Toast.LENGTH_SHORT
        ).show()
    }

}
```

Vista final del patrón MVVM



- kotlin+java
  - com.example.act3_2
    - model
      - Park
      - Parks
    - repository
      - ParkRepository
    - view
      - dialogs
        - MiDialogFragment
      - ui
        - FirstFragment
        - MainActivity
        - SecondFragment
    - viewmodel
      - AdaptadorParques
      - FirstFragmentViewModel
      - ParkViewHolder.kt

12:02

Parques

Parc de Capçalera. València.
Zona verde en el antiguo cauce del Túria, con senderos para caminar o ir en bicicleta.

Parque del Retiro. Madrid
Jardín histórico y parque público.

Parc Güell. Barcelona
Parque público con jardines y elementos arquitectónicos.

12:03

Editar parque

Lunis

Parque infantil

685225458

www.lunis.es

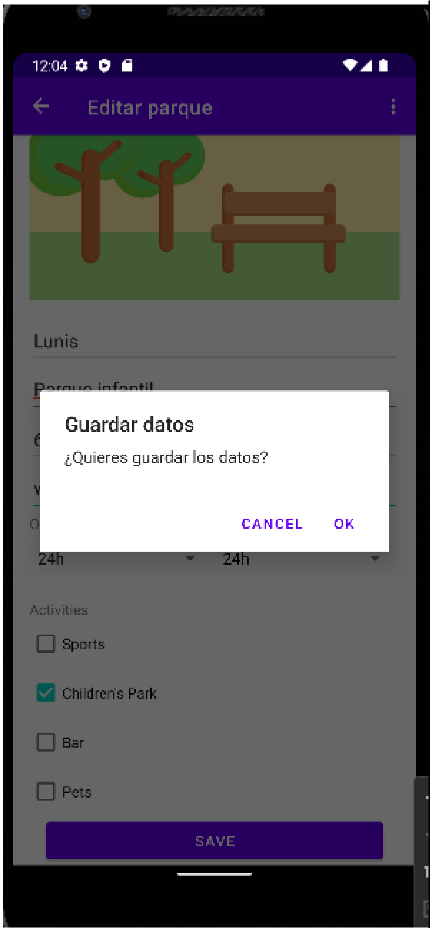
Opening Time: 24h

Closing Time: 24h

Activities

☐ Sports

☑ Children's Park



12:04

Editar parque

Lunis

Parque infantil

Guardar datos

¿Quieres guardar los datos?

CANCEL OK

24h 24h

Activities

☐ Sports

☑ Children's Park

☐ Bar

☐ Pets

SAVE

12:04

Parques

Jardín histórico y parque público.

Parc Güell. Barcelona
Parque público con jardines y elementos arquitectónicos.

Parque de Maria Luisa. Sevilla.
Parque junto a la Plaza de España, con gran cantidad de plantas, aves y zonas de juego.

Lunis
Parque infantil