

## **PMDM. APAC 1.**

SceneMaker en Kotlin



## Continguts

<b>1</b>	<b>Introducció</b>	<b>2</b>
<b>2</b>	<b>Estructura del projecte</b>	<b>2</b>
<b>3</b>	<b>Activitat pràctica</b>	<b>4</b>

## 1 Introducció

En aquesta pràctica anem a començar a treballar amb Kotlin a partir d'un exercici que ja us és conegut: *SceneMaker*, per gestionar una escena que continga diverses figures geomètriques.

El repositori general es troba a Github <https://github.com/joamuran/scene-maker>, i dins ell hi ha una carpeta anomenada *versioKotlin*, amb el codi font de base per a l'exercici.

L'aplicació es troba empaquetada en un projecte Gradle, i podeu executar-la amb:

```
gradle run
```

O fent ús del wrapper:

```
./gradlew run
```

Quan l'executeu, veureu que la interfície consisteix en una línia d'ordres que admet ordres per dibuixar les diferents figures (`rectangle` inicialment), així com llistar-les (`list`), obtenir un llistat d'escenes remotes (`remoteList`) i descarregar-ne alguna (`getFitxer.txt`). També tenim la possibilitat de dibuixar l'escena amb `render`.

Ara per ara, l'aplicació és completament funcional, però només dibuixa figures de tipus *Rectangle*. **La nostra tasca consistirà en generar la resta de figures: Cercle, Linia, Quadrat i Ellipse i fer que l'aplicació funcione amb elles.**

## 2 Estructura del projecte

L'estructura de l'aplicació segueix la d'un projecte amb Gradle d'una única aplicació, trobant-se aquesta al directori `app`, i els fonts a la carpeta `app/src`, seguint l'estructura del paquet `com.ieseljust.kscenemaker`:

```
src/  
├── main  
│   └── kotlin  
│       └── com  
│           └── ieseljust  
│               └── kscenemaker  
│                   ├── App.kt  
│                   ├── Escena.kt  
│                   ├── Figura.kt  
│                   ├── Rectangle.kt  
│                   ├── RemoteManager.kt  
│                   ├── Renderer.kt  
│                   └── Renderizable.kt
```

Veiem què conté cada fitxer:

- **App.kt:** Es tracta de l'objecte de l'aplicació, i conté la funció `main` que la inicia. Els mètodes que més ens interessin són:
  - **StartCli:** Que inicia la CLI, llegint per teclat i parsejant les ordres. Aci **haurem d'implementar el funcionament de l'ordre dimensions**, per modificar la grandària de l'escena (tant quan s'introdueix per la línia d'ordre com quan es troba aquesta en un fitxer descarregat)
  - **addFigura:** Quan es tracta d'una figura, `StartCli` utilitza `addFigura` per afegir la figura a l'escena. **Haurem d'implementar la funcionalitat per a quan s'afigen la resta de figures geomètriques.**
- **Escena.kt:** Conté l'escena de figures geomètriques.
- **Figura.kt:** Superclasse per a les figures.
- **Rectangle.kt:** Subclasse que representa els rectangles.
- **RemoteManager.kt:** Implementa la funcionalitat de connectar-se remotament a un servidor amb una llista de figures de prova (podeu provar a carregar per exemple la figura `ou.txt` que utilitza només rectangles i dibuixar-la). D'aquesta manera, no cal escriure tota l'escena completa per fer proves.
- **Renderer.kt:** Implementa la funcionalitat per dibuixar l'escena en pantalla. Utilitza un Frame de Java Swing per fer-ho.
- **Renderizable.kt:** Interfície implementada per les figures que les obliga a tindre un mètode `render`, per renderitzar aquestes.

### 3 Activitat pràctica

La vostra tasca consistirà en implementar la funcionalitat per a les noves figures.

- Haureu de generar noves classes derivades de *Figura* per a cada tipus de figura nova.
- En el codi que hi ha implementat, principalment a l'objecte *App* haureu d'afegir la funcionalitat corresponent per a que s'admeten les noves figures. Al fitxer *App.kt*, ho teniu indicat amb *ToDos*.

Teniu total llibertat per examinar el codi per veure com està tot implementat (aspectes gràfics, de comunicació en xarxa), però ahi no caldrà modificar res.

Recordeu que teniu l'enunciat per a Java al mateix repositori <https://github.com/joamuran/scenemaker/tree/master/enunciats> i ahi vos conta també com utilitzar algunes funcions gràfiques, que funcionen igual amb Kotlin.