M3 - Programació

POO: Mètodes i operador *this*

Índex

- Mètodes d'una Classe
- Operador ***this***

Mètodes d'una Classe

- Hem vist com declarar classes amb **atributs**.
- Hem vist com afegir **constructors** a una *Classe*.
- Hem vist com afegir i sobreescriure el comportament del mètode **toString**.
- Ara veurem com podem **dotar de funcionalitats als objectes** de la nostra Classe a través de la **declaració dels seus mètodes**.

Mètodes d'una Classe

- Com ja sabem de la programació modular, un **mètode** és un fragment de codi que executa una acció concreta a partir d'unes entrades i que pot retornar una sortida.

- En **Programació Orientada a Objectes**, podem **declarar mètodes** a les nostres estructures de dades complexes *(Classes)* i així **dotar de funcionalitat** als seus objectes instanciats.

- Habitualment, els mètodes d'una Classe, **accedeixen** i **modifiquen** els **valors** dels seus **atributs**.

Mètodes d'una Classe

- Dins d'una classe, podem trobar **diferents tipus de mètodes** en funció del seu propòsit:
  - **Constructors**
  - **Públics**
  - **Privats o *helpers***
  - **Sobreescrits o *override***
  - **Estàtics *o mètodes a nivell de classe***
  - **Abstractes**

Mètodes d'una Classe - Constructors

- En la declaració d'un tipus de dades complex *(una Classe)*, sempre es disposa del **constructor primari** i/o del **constructor per defecte**.

- En funció del llenguatge de programació usat, encara que no els declarem i no escrivim el seu codi, aquests dos constructors ens poden venir donats de sèrie.

Mètodes d'una Classe - Constructors

- El **constructor primari** ens permet inicialitzar els objectes instanciats de la classe donant valor a tots i cadascun dels seus atributs passats per paràmetre a la funció del constructor.

- En canvi, **constructor per defecte**, ens permet inicialitzar els objectes sense passar-li cap valor. Els seus atributs seran informats amb el valor per defecte corresponent al seu tipus de dades. *(Per exemple, el valor per defecte del tipus Int, és 0).*

Mètodes d'una Classe - Constructors

- La resta de constructors, són anomenats **constructors secundaris** i ens permeten definir objectes de la classe de diferents maneres en funció del nombre d'atributs de l'objecte que inicialitzen.

- Per Kotlin, recomanem implementar i fer ús de **constructors secundaris** per tal d'aprendre i tenir coneixement de com estem inicializant els objectes.

Mètodes d'una Classe - Constructors

- Exemple de **constructor secundari** per a la classe *CharacterMarioKart* on només tres dels seus atributs seran informats en el moment d'inicialització dels objectes de la *Classe*:

```
constructor(name: String, speed: Float, acceleration: Float) {
    this.name = name
    this.speed = speed
    this.acceleration = acceleration
}
```

Mètodes d'una Classe - Públics

- Els **mètodes públics** ens permeten dotar de **funcionalitat** als objectes de la classe un cop creats.
- Com que son **públics,** son **accessibles** i es poden fer servir des de qualsevol punt del nostre projecte.
- Son com els **"comandaments"** dels objectes de la *Classe* declarada.

Mètodes d'una Classe - Públics

- Exemples de **mètodes públics** que ens permeten interactuar amb els objectes de la classe *CharacterMarioKart:*

```
...

fun changeWheels(){
    this.traction = 100.0f
}

fun stopKart(){
    this.speed = 0.0f
    this.acceleration = 0.0f
}
...
```

Mètodes d'una Classe - Privats o *helpers*

- Els **mètodes privats** (també anomenats *helpers)*, serveixen per a executar accions sobre els objectes internament des de dins de la Classe.

- Com que són **privats,** només es poden **cridar** des de **dins** d'un altre mètode de **la pròpia classe**.

- Un exemple típic d'ús d'un mètode *helper* és quan volem controlar els valors que pot tenir un atribut concret. Aquest mètode podria ser cridat des del constructor i des del seu setter, però no des de fora de la classe.

Mètodes d'una Classe - Privats o *helpers*

- Exemple d'ús de **mètode privat** a la classe *CharacterMarioKart:*

```
…
fun accelerate(){
    if (this.acceleration > 0) {
        this.acceleration *= 0.1f
        this.calcSpeed(10)
    }
}

private fun calcSpeed(time: Int){
    this.speed = this.acceleration * time
}
…
```

Mètodes d'una Classe - Sobreescrits

- Els **mètodes sobreescrits** (també anomenats *override)*, serveixen per a **redefinir el comportament** d'un mètode **prèviament definit**.

- Això pot ser útil quan volem redefinir el comportament d'accions per defecte tals com "*què passa quan imprimeixo un objecte de la Classe?*" i també en un escenari amb **herència, implementació *d'interfaces*** o de mètodes **abstractes.**

- Haurem de fer ús de l'anotació **override** just a l'inici de la declaració del mètode a sobreescriure.

Mètodes d'una Classe - Sobreescrits

- Exemple de **override** del mètode **toString():**

```
override fun toString(): String {
    return "CharacterMarioKart(name='$name',
                               speed=$speed,
                               acceleration=$acceleration,
                               weight=$weight,
                               handling=$handling,
                               traction=$traction)"
}
```

Mètodes d'una Classe - Estàtics

- Un **mètode estàtic**, és un mètode declarat a **nivell de Classe**. És a dir, **només pot ser executat a través de la Classe** i no a través d'un dels seus objectes.

- Serveixen per a **executar accions estàndard** i **repetitives** que **no** estan **vinculades a un tipus de dades concret**.

- Els mètodes estàtics són usats dins de llibreries de codi tals com la nostra coneguda ***Utilities.kt*** on tots els seus **mètodes són estàtics.** Ja que per usar-los, no hem hagut de crear cap objecte del tipus Utilities.

Mètodes d'una Classe - Estàtics

- Exemple d'ús del **mètode estàtic *readInt*** de la classe **Utilities** dins del Main:

```
import utils.*

fun main(){
    var x: Int = readInt(
        "Tira el dau i escriu un valor entre 1 i 6",
        "Tornar-ho a provar i escriu un valor enter",
        "Escriu un valor entre 1 i 6 !!!",
        1,
        6
    )
}
```

Mètodes d'una Classe - Abstractes

- Un **mètode abstracte**, és un mètode declarat però sense cos definit. No s'especifica el seu comportament.

- Es **declara** que existeix un mètode amb un **nom concret**, però **no s'especifiquen les accions que fa** ja que **no s'escriu el seu codi**.

- Els mètodes abstractes, s'usen **en el context d'herència de classes** i en ***interfícies*** per tal d'obligar a les classes filles a que implementin el seu contingut i així, el seu comportament.

Operador *this*

- En Programació Orientada a Objectes en Kotlin, l'operador ***this*** correspon a una **paraula reservada** del llenguatge que serveix per a **fer referència als atributs** i **mètodes** del **context a on es troba**.

- L'operador ***this***, s'usa dins del codi dels mètodes d'una Classe:
  - Per a **fer referència als atributs** de la **pròpia classe**.
  - Per a **cridar mètodes declarats** dins de la **pròpia classe**.

Operador *this*

- En el cas dels **constructors** serà necessari usar-lo si els paràmetres d'entrada es diuen igual que els atributs de la classe:

```
constructor(name: String, speed: Float, acceleration: Float) {
    name = name
    this.speed = speed
    this.acceleration = acceleration
}
```

- Tot i que no sempre serà necessari fer ús de l'operador ***this***, és recomanable usar-lo sempre que es pugui en la definició de les classes.

Operador *this*

- Exemple en el mètode ***toString()***:

```
override fun toString(): String {
    return "CharacterMarioKart(name=${this.name},
                               speed=${this.speed},
                               acceleration=${this.acceleration},
                               weight=${this.weight},
                               handling=${this.handling},
                               traction=${this.traction}
                               )"
}
```