Kotlin BBL

Igor Lab

Igor Laborie

Expert Web & Java

@ilaborie

igor@monkeypatch.io

<Monkey Patch/>

Pourquoi un nouveau langage ?

- Écrire du code plus sûr
- Facilité la maintenance
- Écrire et Tester plus rapidement
- Résoudre de nouveaux problèmes
- ...

Caractéristiques de Kotlin

- Expressif et pragmatique
- *null-safety* (éviter les NPE), statiquement typé
- Abordable, si on vient de Java
- Inspiré par Java, Scala, C#, Groovy, ...
- Cross-platform

Cible


JVM et Android

JavaScript

Native avec LLVM

Hello World


```
fun main(args: Array<String>) {
    println("Hello World!")
}
```

✨ Utilisez `Alt` + `Shift` + `(Cmd|Ctrl)` + `K` pour convertir une classe Java en Kotlin Ou copiez du code Java dans un fichier Kotlin

Roadmap

I. Water Pouring Problem

II. Live Coding

III. Kotlin dès maintenant

IV. Conclusion

Water Pouring Problem

Verres

8 / 8

0 / 6

1 / 4

Remplir

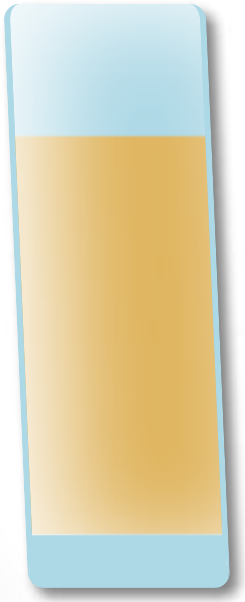
Fill

Vider

Empty

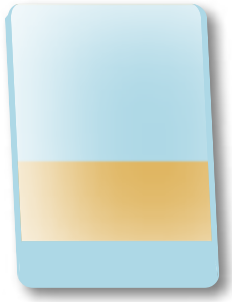
Verser

Pour



6 / 8

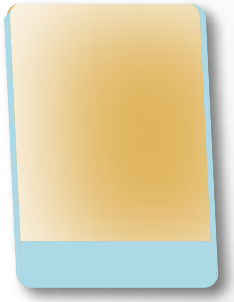
into



2 / 6

2 / 8

,



6 / 6

Démo

Live Coding

Glass & State


```
data class Glass(val capacity: Int, val current: Int = 0) {
    init {
        require(capacity > 0)
        require(current in 0..capacity)
    }

    val isEmpty: Boolean = (current == 0)
    val isFull: Boolean = (current == capacity)
    val remainingVolume: Int  by lazy { capacity - current }
    fun empty(): Glass = copy(current = 0)
    fun fill(): Glass = copy(current = capacity)
    operator fun plus(value: Int): Glass =
        copy(current = (current + value).coerceAtMost(capacity))
    operator fun minus(value: Int): Glass =
        copy(current = (current - value).coerceAtLeast(0))
    override fun toString() = "$current/$capacity"
}

typealias State = List<Glass>
```

Move


```
sealed class Move

data class Empty(val index: Int) : Move()

data class Fill(val index: Int) : Move()

data class Pour(val from: Int, val to: Int) : Move() {
    init {
        require(from ≠ to)
    }
}
```

ℹ️ Avec les `sealed` et les `data class` on peut faire des *Abstract Data Class* Le `sealed` nécessite Kotlin 1.1.

Solver


```
te typealias StateWithHistory = Pair<State, List<Move>>

olve(from: State, to: State): List<Move> {
ailrec fun solveAux(states: List<StateWithHistory>, visitedStates: Set<Stat
   val solution: StateWithHistory? = states.find { (state, _) → state == t
   if (solution ≠ null) { return solution.second }

   val next = states
       .flatMap { (state, history) →
           val moves = state.availableMoves()
           moves.map { move → state.move(move) to (history + move) }
       }
       .filterNot { (state, _) → visitedStates.contains(state) }
   val nextVisited = visitedStates + next.map { it.first }
   return solveAux(next, nextVisited)

eturn solveAux(listOf(from to emptyList()), setOf(from))
```

State::move


```
fun State.move(move: Move): State =
    mapIndexed { index, glass →
        when (move) {
            is Empty → if (index == move.index) glass.empty() else glass
            is Fill  → if (index == move.index) glass.fill() else glass
            is Pour  → when (index) {
                move.from → glass - get(move.to).remainingVolume
                move.to   → glass + get(move.from).current
                else      → glass
            }
        }
    }
```

Kotlin dès maintenant

Serveur

Web et Natif

Conclusion

Pourquoi Kotlin

- Code plus sûr
- Code plus simple
- Interoperable avec Java
- Outillage
- Ecosystème
- Mature
- Simple à apprendre

Liens

- Koans
- Référence
- https://kotlin.link/
- Blog
- Forum
- Slack
- Kotlin Evolution and Enhancement Process