Structured concurrency with Kotlin coroutines

Alessandro Candolini

December 15, 2018

Agenda

```
interface Contract { // mvp contract

    interface View {

    }

    interface Presenter {

    }
}
```

```
interface Contract { // mvp contract example

    interface View {
        fun showLoading()
        fun hideLoading()
        fun showError(error: String)
        fun showResults(items: List<Item>)
    }

    interface Presenter {
        fun onRefresh()
    }
}
```

```
class Presenter : Contract.Presenter {

    override fun onRefresh() = TODO()

}
```

Two-way bindings (with passive view):

- View instance holds a reference to its presenter
- Presenter instance holds a reference to the associated view instance

(Circular dependence; see
`https://www.martinfowler.com/eaaDev/uiArchs.html`)

```
class Presenter : Contract.Presenter {

    override fun onRefresh() {
      view.showLoading()  // <--  view?
    }
}
```

```
class Presenter(
        private val view: Contract.View
) : Contract.Presenter {

    override fun onRefresh() {
      view.showLoading()
    }
}
```

```
interface Contract {

    interface View /* ... */

    interface Presenter {
        fun onBind(view : View) // <---
        fun unbind()            // <---
        fun onRefresh()
    }
}
```

```
class Presenter : Contract.Presenter {

    private var view : Contract.View? = null

    override fun onBind(view: Contract.View) {
        this.view = view
    }
    override fun unbind() {
        this.view = null
    }
    override fun onRefresh() {
        view?.showLoading()
    }
}
```

java.lang.IllegalStateException

```
interface UseCase {

    fun fetch(): List<Item>

}
```

```
class Presenter(
        private val view: Contract.View,
        private val useCase: UseCase
) : Contract.Presenter {

    override fun onRefresh() {
        val items = useCase.fetch()
        view.showItems(items)
    }

    override fun onSubmit() = TODO()

}
```

Questions?