Imperative Reactive Programming: The Observer Pattern

Principles of Reactive Programming

Martin Odersky

The Observer Pattern

The Observer Pattern is widely used when views need to react to changes in a model.

Some variants are also called publish/subscribe, model/view/(controller).

A Publisher Trait

```
trait Publisher {

  private var subscribers: Set[Subscriber] = Set()

  def subscribe(subscriber: Subscriber): Unit =
    subscribers += subscriber

  def unsubscribe(subscriber: Subscriber): Unit =
    subscribers -= subscriber

  def publish(): Unit =
    subscribers.foreach(_.handler(this))
}
```

A Subscriber Trait

```
trait Subscriber {
  def handler(pub: Publisher)
}
```

Observing Bank Accounts

Let's make `BankAccount` a `Publisher`:

```
class BankAccount extends Publisher {
  private var balance = 0

  def deposit(amount: Int): Unit =
    if (amount > 0) {
      balance = balance + amount

    }
  def withdraw(amount: Int): Unit =
    if (0 < amount && amount <= balance) {
      balance = balance - amount

    } else throw new Error("insufficient funds")
}
```

Observing Bank Accounts

Let's make `BankAccount` a `Publisher`:

```
class BankAccount extends Publisher {
  private var balance = 0
  def currentBalance: Int = balance        // <---
  def deposit(amount: Int): Unit =
    if (amount > 0) {
      balance = balance + amount
      publish()                            // <---
    }
  def withdraw(amount: Int): Unit =
    if (0 < amount && amount <= balance) {
      balance = balance - amount
      publish()                            // <---
    } else throw new Error("insufficient funds")
}
```

An Observer

A `Subscriber` to maintain the total balance of a list of accounts:

```
class Consolidator(observed: List[BankAccount]) extends Subscriber {
  private var total: Int = sum()

  private def sum() =
    observed.map(_.currentBalance).sum

  def handler(pub: Publisher) = sum()

  def totalBalance = total
}
```

Observer Pattern, The Good

- Decouples views from state
- Allows to have a varying number of views of a given state
- Simple to set up

Observer Pattern, The Bad

- Forces imperative style, since handlers are `Unit`-typed
- Many moving parts that need to be co-ordinated
- Concurrency makes things more complicated
- Views are still tightly bound to one state; view update happens immediately.

To quantify (Adobe presentation from 2008):

- $1/3^{rd}$ of the code in Adobe's desktop applications is devoted to event handling.
- $1/2$ of the bugs are found in this code.

How to Improve?

The rest of this course will explore different ways in which we can improve on the imperative view of reactive programming embodied in the observer pattern..

This week:	Functional Reactive Programming.
Next two weeks:	Abstracting over events and eventstreams with Futures and Observables.
Last three weeks:	Handling concurrency with Actors.