Scala Developers Barcelona (#scalabcn)

Welcome Scala Newbies Autumn Edition (sept 2013)

sponsored by

special appearance

Jordi Pradel (@agile_jordi)
Ignasi Marimon-Clos (@ignasi35)

thanks!

about me

(Jordi to fill this one in)

about me

n. /iŋ'nazi/

1) ~~problem solver, Garbage Collector, mostly~~ ~~java~~, learning scala, some agile

2) kayaker

3) under construction

4) wears glasses

about you

Start with why

This is your (jee) life

```
public class PurchaseProcessorFactory {

    public PurchaseProcessor forPoyPol() {
        return new PurchaseProcessor() {
            @Override
            public void process(Purchase t) {
                // process t using PoyPol system
            }
        };
    }

    public PurchaseProcessor forViso() {
        return new PurchaseProcessor() {
            @Override
            public void process(Purchase t) {
                // process t using Viso
            }
        };
    }
}
```

This is your (jee) life

```java
package com.meetup.java;

public class Item {

	private final String _productId;
	private final int _rupees;

	public Item(String _productId, int _rupees) {
		super();
		this._productId = _productId;
		this._rupees = _rupees;
	}

	@Override
	public String toString() {
		return "Item [_productId=" + _productId + ", _rupees=" + _rupees + "]";
	}

	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result
				+ ((_productId == null) ? 0 : _productId.hashCode());
		result = prime * result + _rupees;
		return result;
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Item other = (Item) obj;
		if (_productId == null) {
			if (other._productId != null)
				return false;
		} else if (!_productId.equals(other._productId))
```

```java
package com.meetup.java;

import java.util.Collections;
import java.util.List;

public class Purchase {

	private final List<Item> _items;

	public Purchase(List<Item> items) {
		super();
		_items = items;
	}

	public List<Item> getItems() {
		return Collections.unmodifiableList(_items);
	}

	@Override
	public String toString() {
		return "Purchase [_items=" + _items + "]";
	}

	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((_items == null) ? 0 : _items.hashCode());
		return result;
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
```

What's wrong there?

- class per file (sort of)
- unnecessary redefinition of stuff
- duplicate intent
  - (DRY is not only about copy/paste)
- unmaintainability
- JAVA

Let's redo that

```
package com.meetup.scala

case class Item(productId : String, rupees : Int)
case class Purchase(items : List[Item])

trait Processor[T] {
  def process(t : T) : Unit
}
trait PurchaseProcessor extends Processor[Purchase]

object PurchaseProcessor {

  val forPoyPol = new PurchaseProcessor {
    def process(p : Purchase) = println(p) // do real stuff here
  }
  val forViso = new PurchaseProcessor {
    def process(p : Purchase) = println(p) // do real stuff here
  }

}
```

fits in one slide (and fixes one bug!)

Entering decompiler

(Insert picture of Morpheus holding red pill and blue pill)

case classes

- all boilerplate gone
- no equals/hashCode/toString maintenance
- immutable
- named params
- optional params (not shown)
- pattern matching

pater-WAT ??

Pattern Matching

```
def run(input : Any) = {
  input match {
    case s : String => println(s)
    case i : Int    => println("A number: " + i)
    case _          => println("something else")
  }
}

run("A long time ago in a galaxy far far away...")

run(234)
run(() => 42)
```

```
> run: (input: Any)Unit

> A long time ago in a galaxy
> far far away...
> A number: 234
> something else
```

Worksheet rocks! REPL rocks!

Functions!

```
run(() => 42)       > something else
```

WTF???

- Functions as first class citizens
  - Functional Progamming! Bitches!
- parens + arrow + statements
- ‘return’ keyword is optional
- SOLUTION: A function with no params that simply returns ’42’

Functional Programming

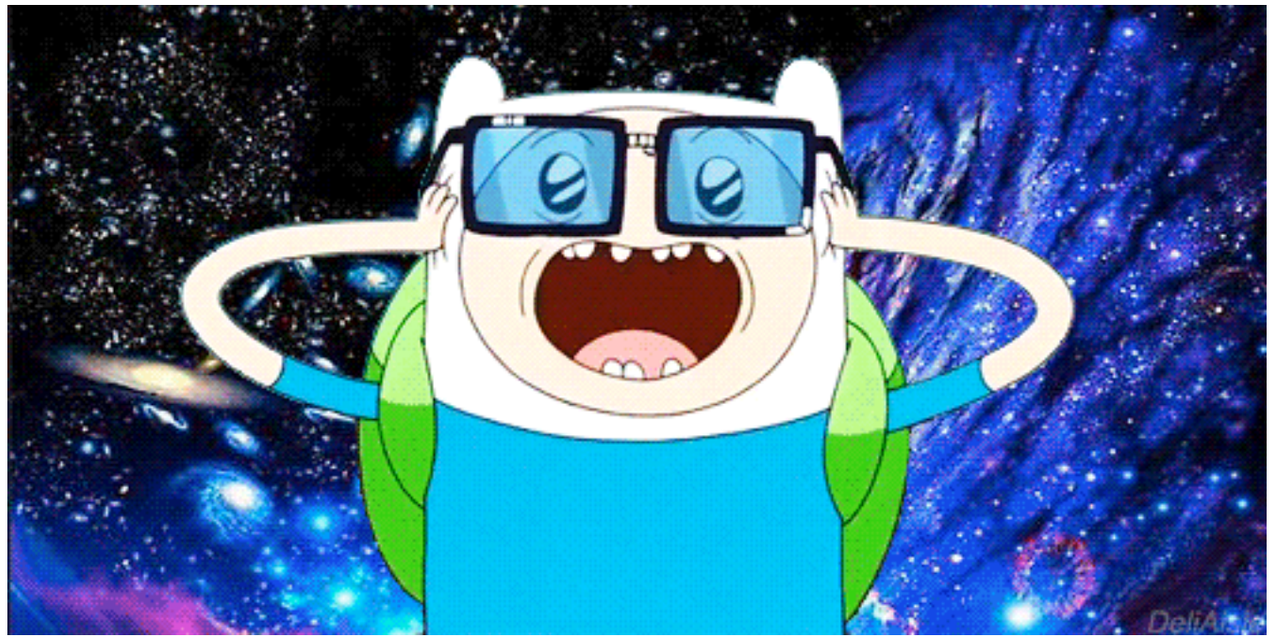
- referential transparency
- immutability (see case classes)
- no side effects
- separate data from behavior
- monads *

* We don't know what monads are but a talk about FP requires using the word monad

Back to coding!

so far

- conciseness
- case classes
- traits
- *multi-hierarchy*
- pattern matching
- Functions
- immutability
- monads *
- spaces/parens/dots
- return keyword



Yak shaving

- Any seemingly pointless activity which is actually necessary to solve a problem which solves a problem which, several levels of recursion later, solves the real problem you're working on.

http://www.urbandictionary.com/define.php?term=yak%20shaving

conclusions

- A lot of syntactic sugar
- Many small features but all related
- functional!
- object oriented (unlike clojure/erlang/...)
- typesafe without the pain
- Classes, Objects, Companions, Traits...

other stuff

- Macros
- Currying, partially applied functions, ...
- Structural Typing (see? we _do_ have duck typing!)
- implicits
- Value classes
- Lazy evaluation
- Tail recursion
- ...

want more?

- Intro to scala: http://scalacamp.pl/intro/#/start
- Twitter's Scala School: http://twitter.github.io/scala_school/
- 99 problems in scala: http://aperiodic.net/phil/scala/s-99/
- Scala Puzzles: http://scalapuzzlers.com/
- "Programming in Scala" (1st ed free online: http://www.artima.com/pins1ed/)

- Scala Developers Barcelona Meetup

thanks!

Oh! And we know what monads are. ;-)