JetBRAINS

The Kotlin Programming Language

Andrey Breslav
Dmitry Jemerov

JetBRAINS

What is Kotlin?

- Statically typed
- JVM-targeted
- general-purpose
- programming language
- developed by JetBrains

- Docs available today
- Beta planned for the end of 2011

JetBRAINS

What is Kotlin? (II)

- Number of research papers we plan to publish related to Kotlin?
  - Zero
  - Or close to that…

JetBRAINS

Outline

- Motivation
- Basic constructs walk-through
- Higher-order functions
  - Function literals
  - Inline functions
  - Type-safe Groovy-style builders
  - Non-local returns
- **Workshop**
  - Classes, Mixins and First-class delegation
  - Generics: Variance annotations and Type projections
  - Class objects
  - Pattern matching

JetBRAINS

Motivation

- Why a new language?
  - The existing ones do not fit with our needs
  - And we had a close look at many of them
- Design goals
  - Full **Java interoperability**
  - **Compiles** as fast as **Java**
  - **Safer** than **Java**
  - More **concise** than **Java**
  - Way **simpler** than **Scala**

JetBRAINS

Feature list

- Features:
  - Higher-order functions
  - Properties
  - Mixins and First-class delegation
  - Extension functions
  - Static nullability checking
  - Automatic casts
  - Reified generics
  - Declaration-site variance
  - Modules and Build infrastructure (fighting the "Jar hell")
  - Inline-functions (zero-overhead closures)
  - Pattern matching
  - …
- Full-featured IDE by JetBrains from the very beginning

JetBRAINS

Basic constructs walk-through

- IDE demo

JetBRAINS

Function types and literals

- Functions

```
fun f(p : Int) : String { return p.toString() }
```

- Function types

```
fun (p : Int) : String
fun (Int) : String
```

- Function literals

```
{(p : Int) : String => p.toString()}
{(p : Int) => p.toString()}
{p => p.toString()}
```

JetBRAINS

Higher-order functions

```
fun filter<T>(
      c : Iterable<T>,
      f : fun (T) : Boolean
    ) : Iterable<T>
val list = list("a", "ab", "abc", "")
filter(list, {s => s.length() < 3})
```

- yields ["a", "ab", ""]
- Convention: last function literal argument

```
filter(list) {s => s.length() < 3}
```

- Convention: one-parameter function literal

```
filter(list) {it.length() < 3}
```

JetBRAINS

Lock example (I)

```
myLock.lock()
try {
   // Do something
} finally {
   myLock.unlock()
}
```

JetBRAINS

Lock example (II)

```
lock (myLock) {
  // Do something
}
```

```
fun lock(
  theLock : Lock,
  body : fun() : Unit
)
```

JetBRAINS

Implementation

- **General**: works everywhere, but costs something
  - Inner classes
  - Method handles
- **Special**: may not work in some cases, but costs nothing
  - Inline functions
- **Kotlin** features **both** general and special implementations

JetBRAINS

Lock example (III)

```
inline fun lock(
  theLock : Lock,
  body : fun() : Unit
) {
  myLock.lock()
  try {
    body()
  } finally {
    myLock.unlock()
  }
}
```

JetBRAINS

Extension function literals

- Extension functions

```
fun Int.f(p : Int) : String { return "..." }
```

- Extension function types

```
fun Int.(p : Int) : String
fun Int.(Int) : String
```

- Extension function literals

```
{Int.(p : Int) : String => "..."}
{Int.(p : Int) => "..."}
{Int.(p) => "..."}
```

JetBRAINS

HTML example (I)

- Function definition

```
fun html(init : fun HTML.() : Unit) : HTML {
  val html = HTML()
  html.init()
  return html
}
```

- Usage

```
html {
  this.addMeta(
    httpEquiv="content-type",
    content="text/html;charset=UTF-8")
}
```

JetBRAINS

HTML example (II)

- Function definition

```
fun html(init : fun HTML.() : Unit) : HTML {
  val html = HTML()
  html.init()
  return html
}
```

- Usage

```
html {
  addMeta(
    httpEquiv="content-type",
    content="text/html;charset=UTF-8")
}
```

JetBRAINS

Builders in Groovy

```
html {
  head {
    title "XML encoding with Groovy"
  }
  body {
    h1 "XML encoding with Groovy"
    p "this format can be used as an alternative markup to XML"

    /* an element with attributes and text content */
    ahref:'http://groovy.codehaus.org' ["Groovy"]
}
```

JetBRAINS

Builders in Kotlin

```
html {
  head {
    title { +"XML encoding with Kotlin" }
  }
  body {
    h1 { +"XML encoding with Kotlin" }
    p { +"this format can be used as an alternative markup to XML" }

    /* an element with attributes and text content */
    a (href="http://jetbrains.com/kotlin") { +"Kotlin" }
}
```

- The big difference: the **Kotlin** version is **statically type-checked**

JetBRAINS

Builders in Kotlin: Implementation (I)

```
abstract class Tag(val name : String) : Element {
  val children = ArrayList<Element>()
  val attributes = HashMap<String, String>()
}

abstract class TagWithText(name : String) : Tag(name) {
  fun String.plus() {
    children.add(TextElement(this))
  }
}

class HTML() : TagWithText("html") {
  fun head(init : fun Head.() : Unit) { … }
  fun body(init : fun Body.() : Unit) { … }
}
```

JetBRAINS

Builders in Kotlin: Implementation (II)

```
fun html(init : fun HTML.() : Unit) : HTML {
  val html = HTML()
  html.init()
  return html
}


class HTML() : TagWithText("html") {


  fun head(init : fun Head.() : Unit) {
    val head = Head()
    head.init()
    children.add(head)
  }


}
```

JetBRAINS

Builders in Kotlin: Implementation (III)

```
 a (href="http://jetbrains.com/kotlin") { +"Kotlin" }


class BodyTag(name : String) : TagWithText(name) {

  fun a(href : String, init : fun A.() : Unit) : A {
    val a = A()
    a.init()
    a.attributes["href"] = href
    children.add(a)
  }

}
```

JetBRAINS

Foreach example (I)

```
inline fun <T> Iterable<T>.foreach(
  body : fun(T) : Unit
) {
  for (item in this)
    body(item)
}
```

Example usage:

```
  list map {it.length() > 2} foreach {
    print(it)
  }
```

JetBRAINS

Foreach example (II)

```
fun hasZero(list : List<Int>) : Boolean {
  // A call to an inline function
  list.foreach {
    if (it == 0)
      return true // Non-local return
  }
  return false
}
```

- Unqualified `return` always returns from a *named* function

JetBRAINS

Qualified returns

- Function literals may be marked with labels:

```
@label {x => ...}
```

- To make a local return, use qualified form:

```
@label { x =>
    ...
    return@label
    ...
}
```

JetBRAINS

Labels, Break and Continue

```
@outer for (x in list1) {
  for (y in list2) {
    if (...) {
      // Breaks the inner loop
      break
    }
    if (...) {
      // Breaks the outer loop
      break@outer
    }
  }
}
```

JetBRAINS

Breaks in foreach()

```
@outer list1.foreach { x =>
  list2.foreach { y =>
    if (...) {
      // Breaks the inner loop
      break
    }
    if (...) {
      // Breaks the outer loop
      break@outer
    }
  }
}
```

JetBRAINS

Breakable `foreach()`

```
inline fun <T> Iterable<T>.foreach(
  body : breakable fun(T) : Unit
) {
  @@ for (item in this) {
    // A break from body() breaks the loop
    body(item)
  }
}
```

JetBRAINS

Resources

- http://jetbrains.com/kotlin
- http://blog.jetbrains.com/kotlin
- @project_kotlin
- @intelliyole
- @abreslav

JetBRAINS

The Kotlin Programming Language

Andrey Breslav
Dmitry Jemerov