Scala plugin for IntelliJ IDEA

Pavel Fatin

Alexander Podkhalyuzin

Scalathon, 2011

Scala IntelliJIDEA JetBRAINS

Thank you!

Agenda

IntelliJ Scala Plugin

IntelliJ Scala Plugin

- Why do we need it

Agenda

IntelliJ Scala Plugin

- Why do we need it
- What's inside

Agenda

IntelliJ Scala Plugin

- Why do we need it
- What's inside
- How it's made

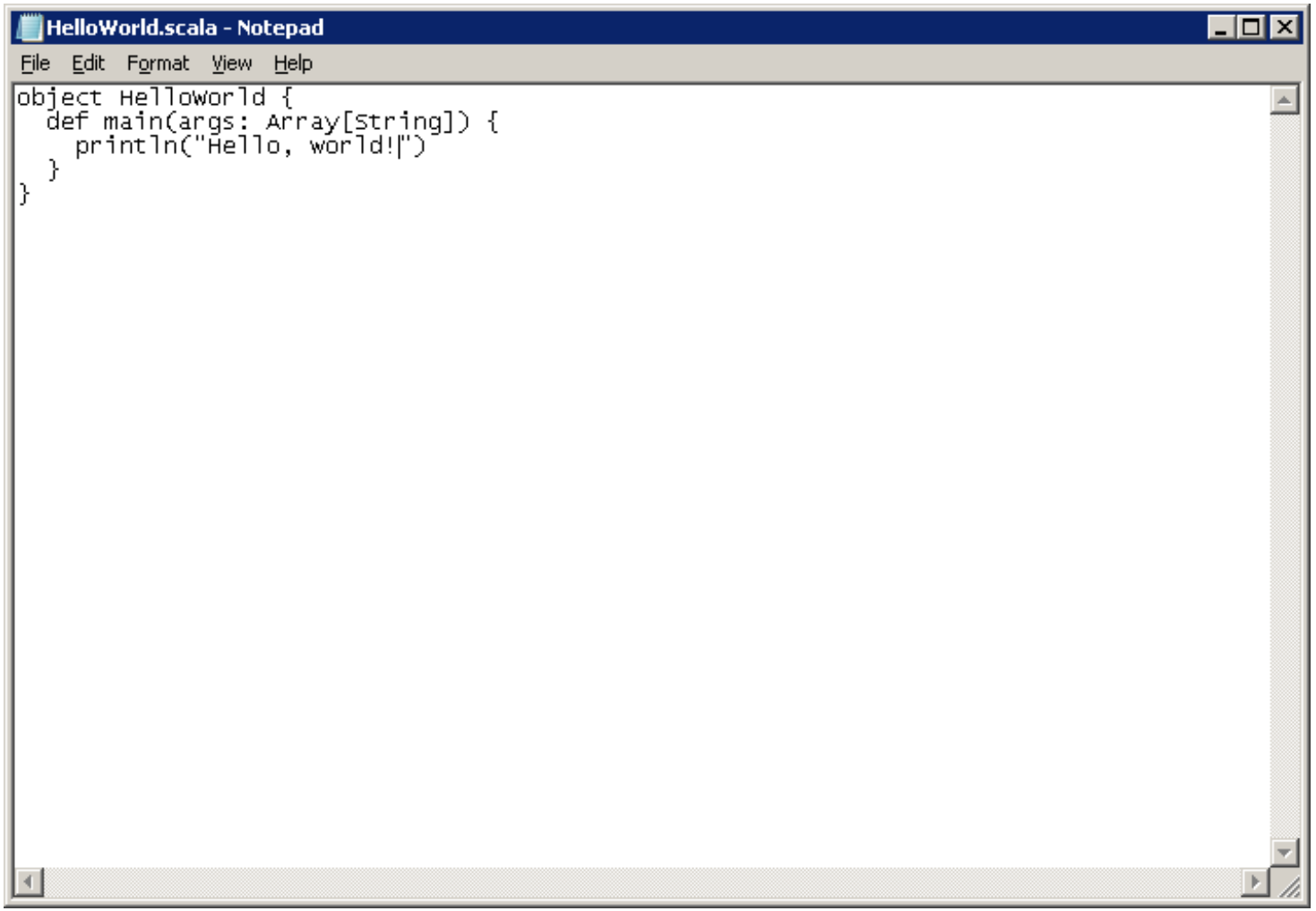
Agenda

IntelliJ Scala Plugin

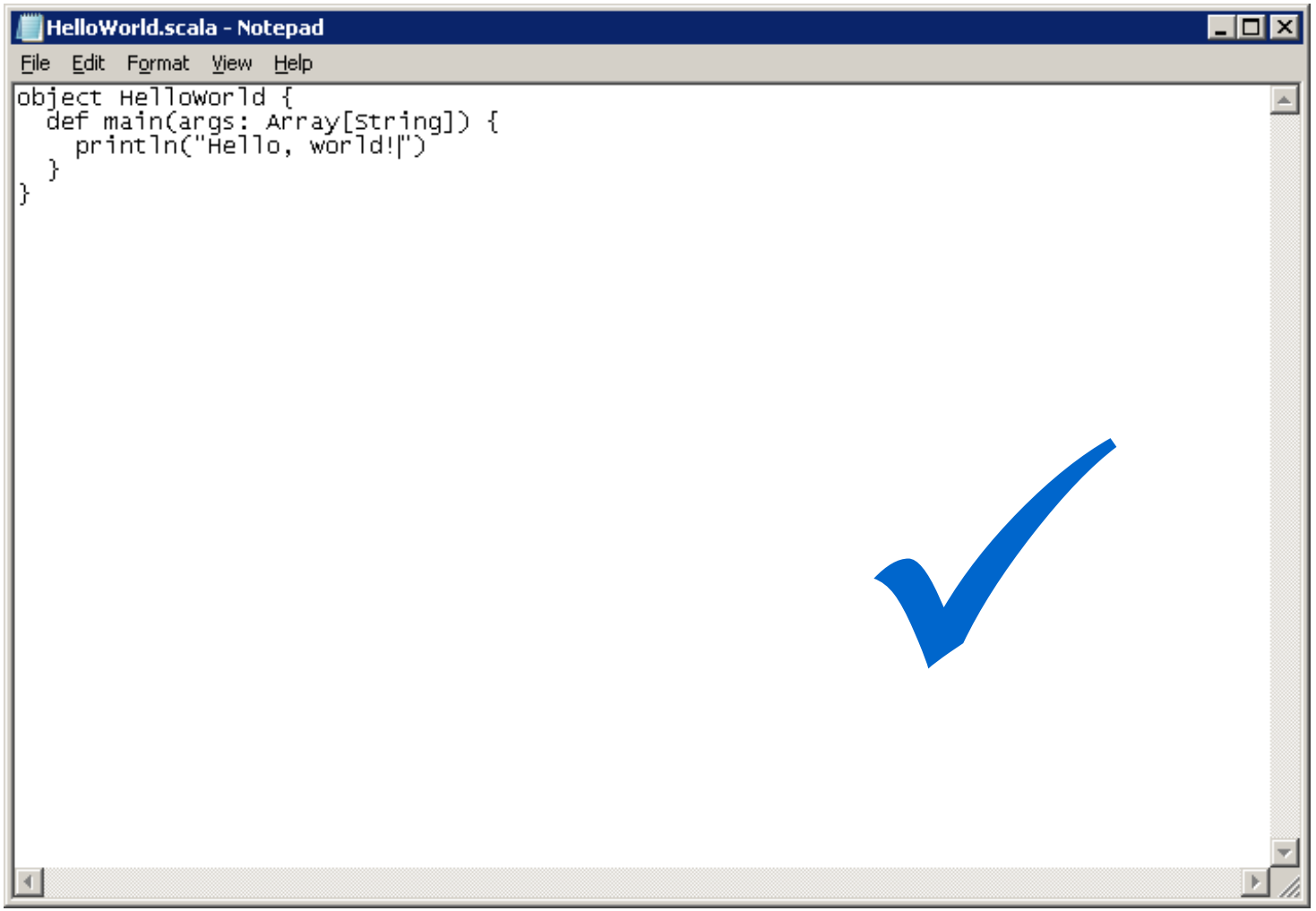
- Why do we need it
- What's inside
- How it's made
- How to contribute

Do we need an IDE for Scala?

Can't we just use Notepad?



Can't we just use Notepad?



Is it always the right tool?

Typers.scala - Notepad

File Edit Format View Help

```
protected def typedExistentialTypeTree(tree: ExistentialTypeTree, mode: Int): Tree =
  for (wc <- tree.whereClauses)
    if (wc.symbol == NoSymbol) { namer.enterSym(wc); wc.symbol setFlag EXISTENTIAL }
    else context.scope enter wc.symbol
  val whereClauses1 = typedStats(tree.whereClauses, context.owner)
  for (vd @ ValDef(_, _, _, _) <- tree.whereClauses)
    if (vd.symbol.tpe.isVolatile)
      error(vd.pos, "illegal abstraction from value with volatile type "+vd.symbol.tp
  val tpt1 = typedType(tree.tpt, mode)
  val (typeParams, tpe) = existentialTransform(tree.whereClauses map (_.symbol), tpt1
  //println(tpe + ": " + tpe.getClass )
  TypeTree(ExistentialType(typeParams, tpe)) setOriginal tree
}

// lifted out of typed1 because it's needed in typedImplicit0
protected def typedTypeApply(tree: Tree, mode: Int, fun: Tree, args: List[Tree]): Tre
  case OverloadedType(pre, alts) =>
    inferPolyAlternatives(fun, args map (_.tpe))
    val tparams = fun.symbol.typeParams //@M TODO: fun.symbol.info.typeParams ? (as i
    val args1 = if (sameLength(args, tparams)) {
      //@M: in case TypeApply we can't check the kind-arities of the type arguments,
      // as we don't know which alternative to choose... here we do
      map2Conserve(args, tparams) {
        //@M! the polytype denotes the expected kind
        (arg, tparam) => typedHigherKindedType(arg, mode, polyType(tparam.typeParams,
      }
    } else // @M: there's probably something wrong when args.length != tparams.length
     // Martin, I'm using fake trees, because, if you use args or arg.map(typedType),
     // inferPolyAlternatives loops...  -- I have no idea why :-(
     // ...actually this was looping anyway, see bug #278.
      return errorTree(fun, "wrong number of type parameters for "+treeSymTypeMsg(fun
```

Rationale

What makes us more productive?

What makes us more productive?

- Highlighting

What makes us more productive?

- Highlighting
- Inspections

What makes us more productive?

- Highlighting
- Inspections
- Code formatting

What makes us more productive?

- Highlighting
- Inspections
- Code formatting
- Auto-completion

What makes us more productive?

- Highlighting
- Inspections
- Code formatting
- Auto-completion
- Information look-up

What makes us more productive?

- Highlighting
- Inspections
- Code formatting
- Auto-completion
- Information look-up
- Refactoring

What makes us more productive?

- Highlighting
- Inspections
- Code formatting
- Auto-completion
- Information look-up
- Refactoring
- Integration

What's already available?

Syntax highlighting

```
import io.Source

object Applicatin {
  // Regex pattern
  val Capitalized = "([A-Z])\\w+".r

  // TODO split method
  def main(args: Array[String]) {
    println(args.headOption.getOrElse(readMessage()))

    for (Capitalized(letter) <- Source.fromFile("lines.txt").getLines()) {
      println(letter)
    }
  }

  def readMessage() = {
    Source.fromInputStream(System.in).getLine(0)
  }
}
```

Syntax highlighting

```
import io.Source

object Applicatin {
  // Regex pattern
  val Capitalized = "([A-Z])\\w+".r

  // TODO split method
  def main(args: Array[String]) {
    println(args.headOption.getOrElse(readMessage()))

    for (Capitalized(letter) <- Source.fromFile("lines.txt").getLines()) {
      println(letter)
    }
  }

  def readMessage() = {
    Source.fromInputStream(System.in).getLine(0)
  }
}
```

Implicit conversion

Syntax highlighting

```
import io.Source

object Applicatin {
  // Regex pattern
  val Capitalized = "([A-Z])\\w+".r

  // TODO split method
  def main(args: Array[String]) {
    println(args.headOption.getOrElse(readMessage()))

    for (Capitalized(letter) <- Source.fromFile("lines.txt").getLines()) {
      println(letter)
    }
  }

  def readMessage() = {
    Source.fromInputStream(System.in).getLine(0)
  }
}
```

By-name argument

Implicit conversion

Syntax highlighting

```
import io.Source

object Applicatin {
  // Regex pattern
  val Capitalized = "([A-Z])\\w+".r

  // TODO split method
  def main(args: Array[String]) {
    println(args.headOption.getOrElse(readMessage()))

    for (Capitalized(letter) <- Source.fromFile("lines.txt").getLines()) {
      println(letter)
    }
  }

  def readMessage() = {
    Source.fromInputStream(System.in).getLine(0)
  }
}
```

By-name argument

Implicit conversion

Deprecated method

Syntax highlighting

```
import io.Source

object Applicatin {
  // Regex pattern
  val Capitalized = "([A-Z])\\w+".r

  // TODO split method
  def main(args: Array[String]) {
    println(args.headOption.getOrElse(readMessage()))

    for (Capitalized(letter) <- Source.fromFile("lines.txt").getLines()) {
      println(letter)
    }
  }

  def readMessage() = {
    Source.fromInputStream(System.in).getLine(0)
  }
}
```

Injected language

By-name argument

Implicit conversion

Deprecated method

Error highlighting

Error highlighting

- Ordinary:

```
val a = 1; val a = 2 // a is already defined as value a

val b = 1; b = 2 // reassignment to val

def f(p: Int) {}; f(1, 2) // too many arguments for method f: (p: Int)Unit
```

Error highlighting

- Ordinary:

```
val a = 1; val a = 2 // a is already defined as value a

val b = 1; b = 2 // reassignment to val

def f(p: Int) {}; f(1, 2) // too many arguments for method f: (p: Int)Unit
```

- Type-aware:

```
val a: Int = "foo" // type mismatch; found: String, required: Int

def f(p: Int) {}; f("foo") // type mismatch; found: String, required: Int

123.substring(1) // value substring is not a member of Int
```

Error highlighting

- Ordinary:

```
val a = 1; val a = 2 // a is already defined as value a

val b = 1; b = 2 // reassignment to val

def f(p: Int) {}; f(1, 2) // too many arguments for method f: (p: Int)Unit
```

- Type-aware:

```
val a: Int = "foo" // type mismatch; found: String, required: Int

def f(p: Int) {}; f("foo") // type mismatch; found: String, required: Int

123.substring(1) // value substring is not a member of Int
```

- Quick-fixes:

```
class Runner extends Runnable
```

Add 'abstract' modifier
Implement methods

Inspections

Inspections

- Unused symbol

Inspections

- Unused symbol
- Variable could be value

Inspections

- Unused symbol
- Variable could be value
- Redundant return

Inspections

- Unused symbol
- Variable could be value
- Redundant return
- Syntactic sugar

Inspections

- Unused symbol
- Variable could be value
- Redundant return
- Syntactic sugar
- Method signature inspections

Language injection

Language injection

- Pattern-based:

```
hibernate.createQuery("select name from Person where name like '%neo%'")
```

Language injection

- Pattern-based:

```
hibernate.createQuery("select name from Person where name like '%neo%'")
```

- Via parameter annotation:

```
def compile(@Language("RegExp") pattern: String) {}
compile("[a-c]{1,3}(\\w+)]")
```

Language injection

- Pattern-based:

```
hibernate.createQuery("select name from Person where name like '%neo%'")
```

- Via parameter annotation:

```
def compile(@Language("RegExp") pattern: String) {}
compile("[a-c]{1,3}(\\w+)]")
```

- Via value annotation:

```
@Language("HTML")
val html = """<html><body title="Content">
                        <style>p { border: solid; }</style>Foo</body></html>"""
```

Dark color schemes

```
import io.Source

object Applicatin {
  // Regex pattern
  val Capitalized = "([A-Z])\\w+".r

  // TODO split method
  def main(args: Array[String]) {
    println(args.headOption.getOrElse(readMessage()))

    for (Capitalized(letter) <- Source.fromFile("lines.txt").getLines()) {
      println(letter)
    }
  }

  def readMessage() = {
    Source.fromInputStream(System.in).getLine(0)
  }
}
```

Language-aware editor

Language-aware editor

- Brace balancing:

```
val i = (1 * (2 + (3 * (4 + 5))))
```

Language-aware editor

- Brace balancing:

```
val i = (1 * (2 + (3 * (4 + 5))))
```

- Entities selection:

```
val i = (1 * (2 + (3 * (4 + 5))))
```

Language-aware editor

- Brace balancing:

```
val i = (1 * (2 + (3 * (4 + 5))))
```

- Entities selection:

```
val i = (1 * (2 + (3 * (4 + 5))))
```

- Code folding:

```
def someComplexMethod() {...}
```

Formatting

Formatting

```
  object  Main{ def   main (args:Array [String ] ){
val res=for  (a<-args)yield a. toUpperCase
      println      ("Arguments: "+res.toString   )}}
```

Formatting

```
object  Main{ def   main (args:Array [String ] ){
val res=for  (a<-args)yield a. toUpperCase
      println      ("Arguments: "+res.toString   )}}
```



```
object Main {
  def main(args: Array[String]) {
    val res = for (a <- args) yield a.toUpperCase
    println("Arguments: " + res.toString)
  }
}
```

Code completion

```
List(1, 2, 3).fo
```

Code completion

```
List(1, 2, 3).fo
```

```
foldLeft[B](z: B)(f: (B, Int) => B)        B
foldRight[B](z: B)(f: (Int, B) => B)       B
forall(p: (Int) => Boolean)            Boolean
foreach[B](f: (Int) => B)                 Unit
formatted(fmtstr: String)              String
firstOption                      Option[Int]
```

Information look-up

Information look-up

- Type info

Information look-up

- Type info
- Parameter info

Information look-up

- Type info
- Parameter info
- Documentation look-up

Information look-up

- Type info
- Parameter info
- Documentation look-up
- Definition look-up

Type Info

```
val v = "157".map(_ asDigit).filter(3 <)
        .zipWithIndex.map(p => p._1 + ":" + p._2)
```

Type Info

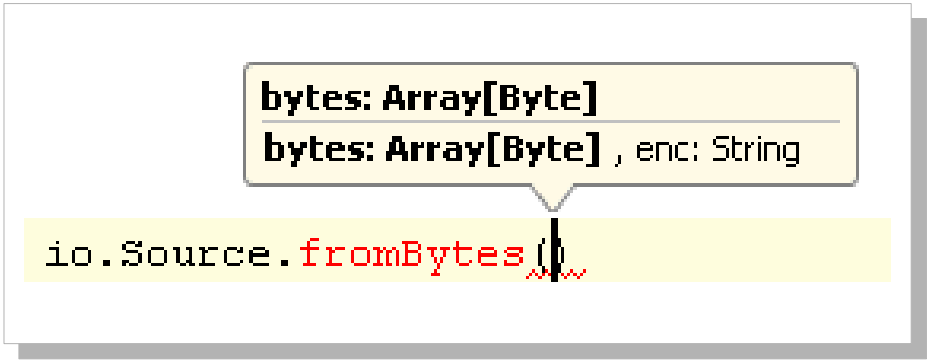
IndexedSeq[String]

```
val v = "157".map(_ asDigit).filter(3 <)
        .zipWithIndex.map(p => p._1 + ":" + p._2)
```

Parameter info

```
io.Source.fromBytes()
```

Parameter info



Documentation look-up

```
List(1, 2, 3).map(_.toString)
```

Documentation look-up

```
List(1, 2, 3).map(_.toString)
```

Documentation for map

scala-library-2.8.0

scala.collection.TraversableLike

```
def map[B, That](f: (A) => B)
(implicit bf: CanBuildFrom[Repr, B, That]): That
```

Builds a new collection by applying a function to all elements of this $coll.

Parameters:

f - the function to apply to each element.

bf - $bfinfo

Returns:

a new collection of type `That` resulting from applying the given function `f` to each element of this $coll and collecting the results.

Definition look-up

```
List(1, 2, 3).map(_.toString)
```

Definition look-up

```
List(1, 2, 3).map(_.toString)
```

Definition of map

TraversableLike.scala scala-library-2.8.0

```
def map[B, That](f: A => B)(implicit bf: CanBuildFrom[Repr, B, That]): That = {
  val b = bf(repr)
  b.sizeHint(this)
  for (x <- this) b += f(x)
  b.result
}
```

Imports management

Imports management

- Import entity on completion

Imports management

- Import entity on completion
- Auto-import unresolved symbol

Imports management

- Import entity on completion
- Auto-import unresolved symbol
- Highlight unused imports

Imports management

- Import entity on completion
- Auto-import unresolved symbol
- Highlight unused imports
- Optimize imports

Imports management

- Import entity on completion
- Auto-import unresolved symbol
- Highlight unused imports
- Optimize imports
- Paste imports

Refactoring

Refactoring

- Rename

Refactoring

- Rename
- Introduce variable

Refactoring

- Rename
- Introduce variable
- Inline variable

Refactoring

- Rename
- Introduce variable
- Inline variable
- Extract method

Refactoring

- Rename
- Introduce variable
- Inline variable
- Extract method
- Introduce parameter

Rename

```
val foo = 5

val width = 30 + foo
val height = 20 + foo

println(width * height)
```

```
val inset = 5

val width = 30 + inset
val height = 20 + inset

println(width * height)
```

Introduce variable

```
val inset = 5

val width = 30 + inset
val height = 20 + inset

println(width * height)
```

```
val inset = 5

val width = 30 + inset
val height = 20 + inset

val area = width * height

println(area)
```

Inline variable

```
val inset = 5

val width = 30 + inset
val height = 20 + inset

val area = width * height

println(area)
```



```
val inset = 5

val width = 30 + inset
val height = 20 + inset

println(width * height)
```

Extract method

```
val inset = 5

val width = 30 + inset
val height = 20 + inset

println(width * height)
```

```
val inset = 5

val width = 30 + inset
val height = 20 + inset

def area = {
  width * height
}

println(area)
```

Integrations

Integrations

- Java

Integrations

- Java
- JUnit / Specs / ScalaTest

Integrations

- Java
- JUnit / Specs / ScalaTest
- Maven

Integrations

- Java
- JUnit / Specs / ScalaTest
- Maven
- Spring

Integrations

- Java
- JUnit / Specs / ScalaTest
- Maven
- Spring
- Lift

Integrations

- Java
- JUnit / Specs / ScalaTest
- Maven
- Spring
- Lift
- SBT

How it's made

(It's not ~~rocket science~~ theoretical physics)

Building the project

Building the project

- Get the latest IDEA EAP build

Building the project

- Get the latest IDEA EAP build
- Download IDEA sources

Building the project

- Get the latest IDEA EAP build
- Download IDEA sources
- Enable DevKit plugin

Building the project

- Get the latest IDEA EAP build
- Download IDEA sources
- Enable DevKit plugin
- Configure a Plugin SDK

Building the project

- Get the latest IDEA EAP build
- Download IDEA sources
- Enable DevKit plugin
- Configure a Plugin SDK
- Clone Scala plugin repository

Building the project

- Get the latest IDEA EAP build
- Download IDEA sources
- Enable DevKit plugin
- Configure a Plugin SDK
- Clone Scala plugin repository
- Open the project in IDEA

Plugging the plugin

Plugging the plugin

- Plugin.xml

```
<extensions defaultExtensionNs="com.intellij">
  <refactoring.moveHandler
      implementation="org.jetbrains.plugins.scala.lang.refactoring.move.ScalaMoveHandler"/>
  <copyPastePreProcessor
      implementation="org.jetbrains.plugins.scala.conversion.copy.StringLiteralProcessor"/>
  <testFramework
      implementation="org.jetbrains.plugins.scala.testingSupport.specs2.Specs2TestFramework"/>

  <intentionAction>
    <className>org.jetbrains.plugins.scala.codeInsight.intention.types.ToggleTypeAnnotation</className>
  </intentionAction>
```

Plugging the plugin

- Plugin.xml

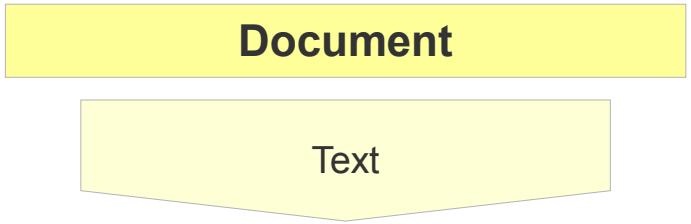
```
<extensions defaultExtensionNs="com.intellij">
  <refactoring.moveHandler
      implementation="org.jetbrains.plugins.scala.lang.refactoring.move.ScalaMoveHandler"/>
  <copyPastePreProcessor
      implementation="org.jetbrains.plugins.scala.conversion.copy.StringLiteralProcessor"/>
  <testFramework
      implementation="org.jetbrains.plugins.scala.testingSupport.specs2.Specs2TestFramework"/>

  <intentionAction>
    <className>org.jetbrains.plugins.scala.codeInsight.intention.types.ToggleTypeAnnotation</className>
  </intentionAction>
```

- OpenAPI

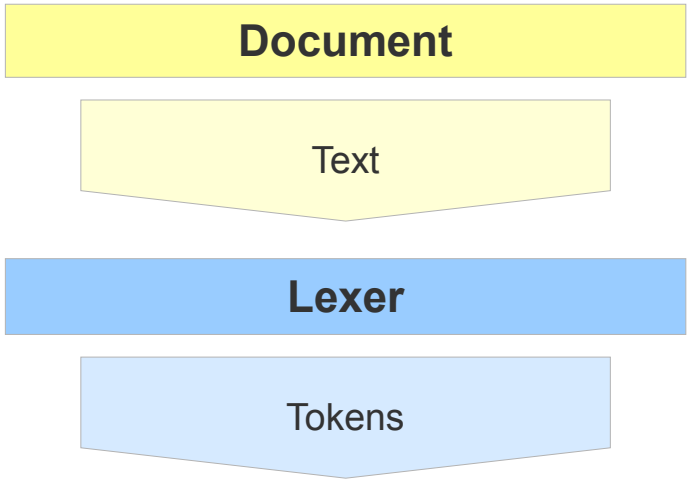
```
import com.intellij.openapi.project._

ProjectManager.getInstance().getOpenProjects
```

Code processing

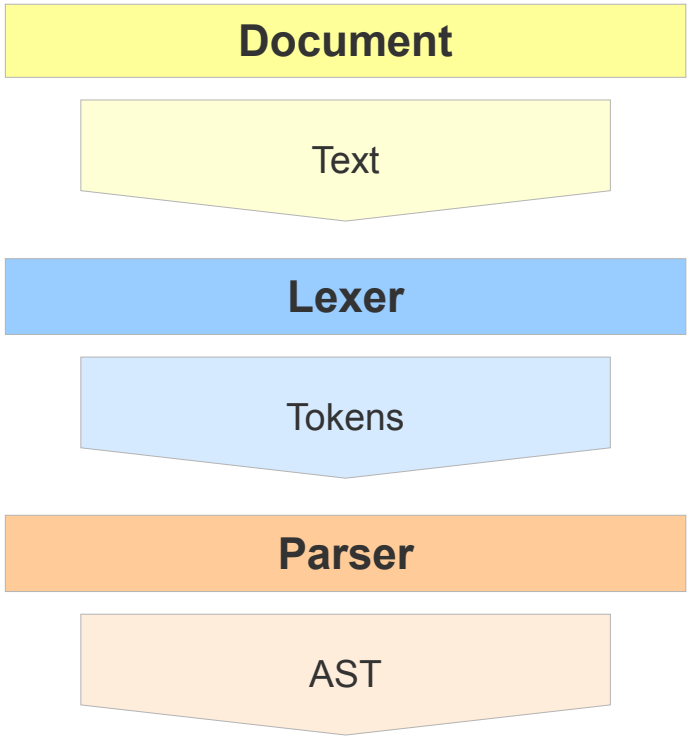
Code processing



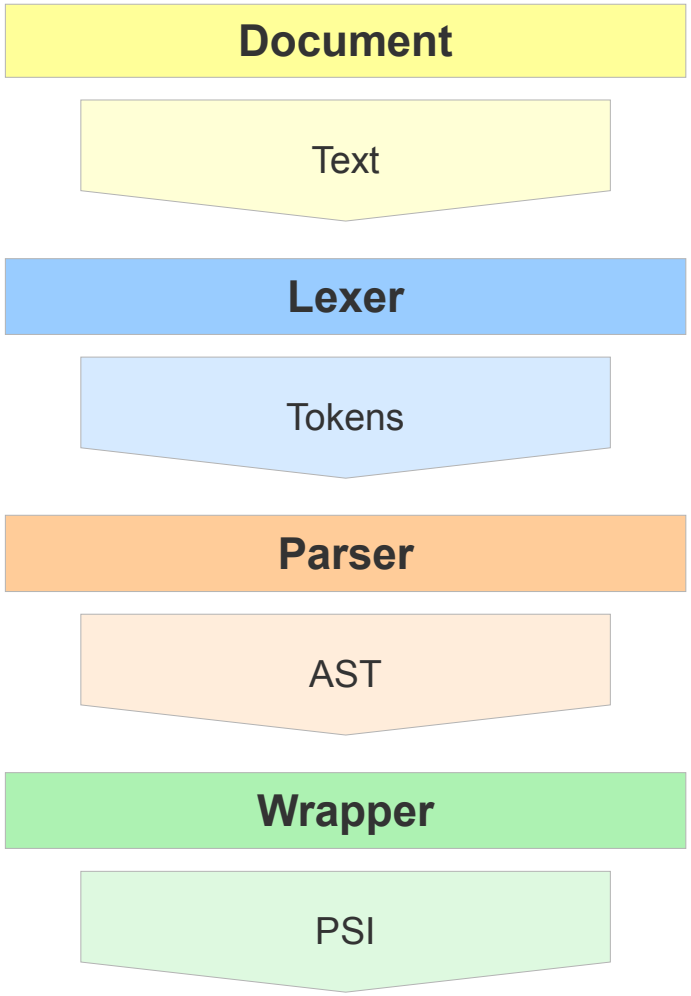
Code processing



Code processing



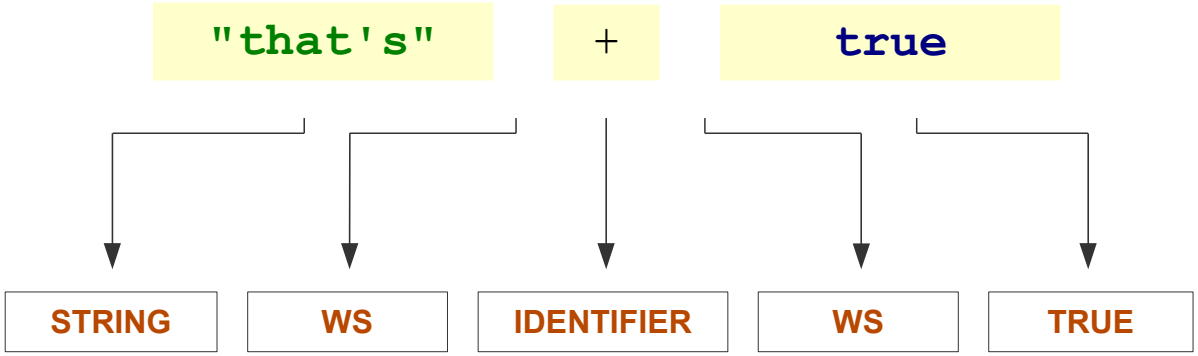
Code processing



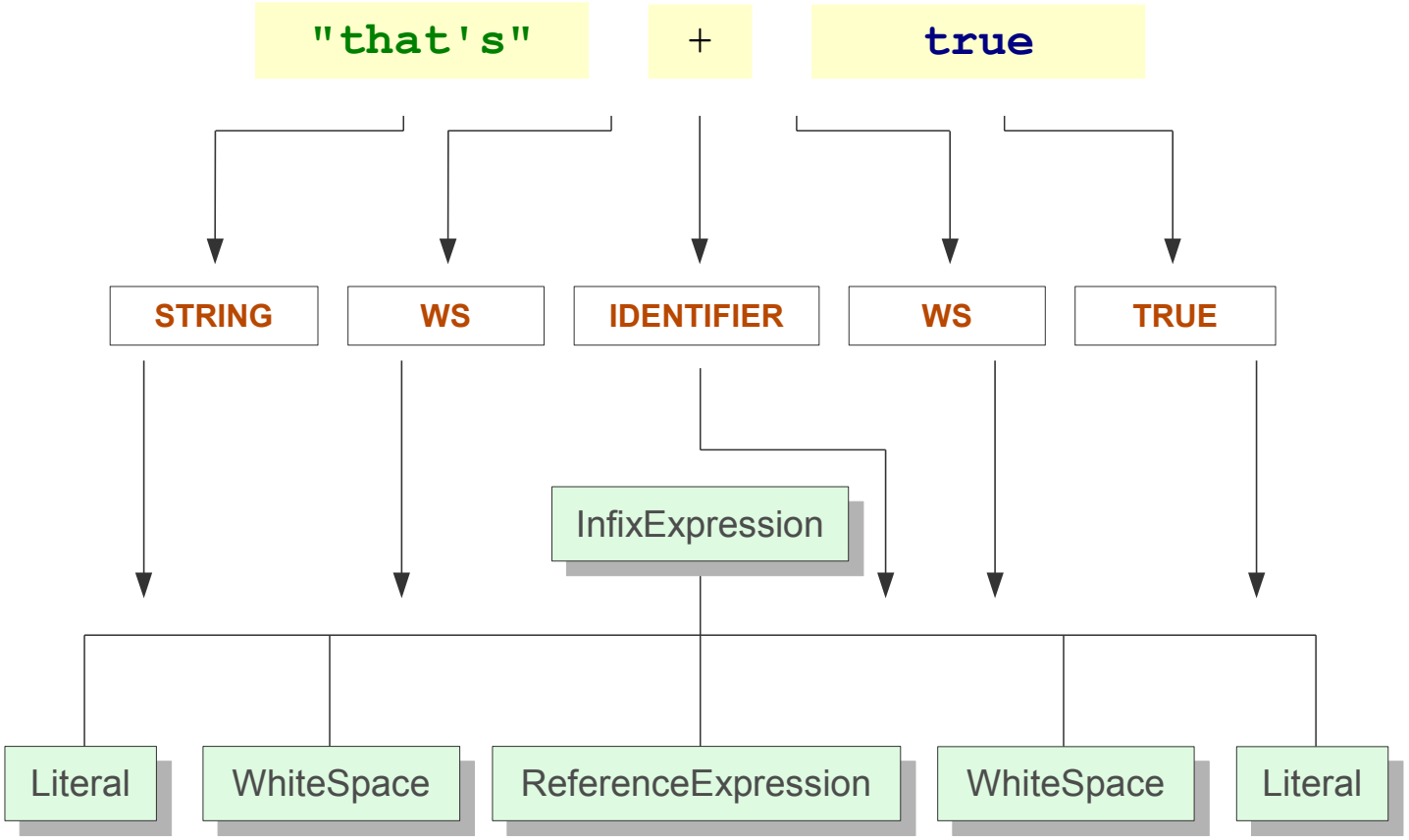
Processing example

```
"that's"  +  true
```

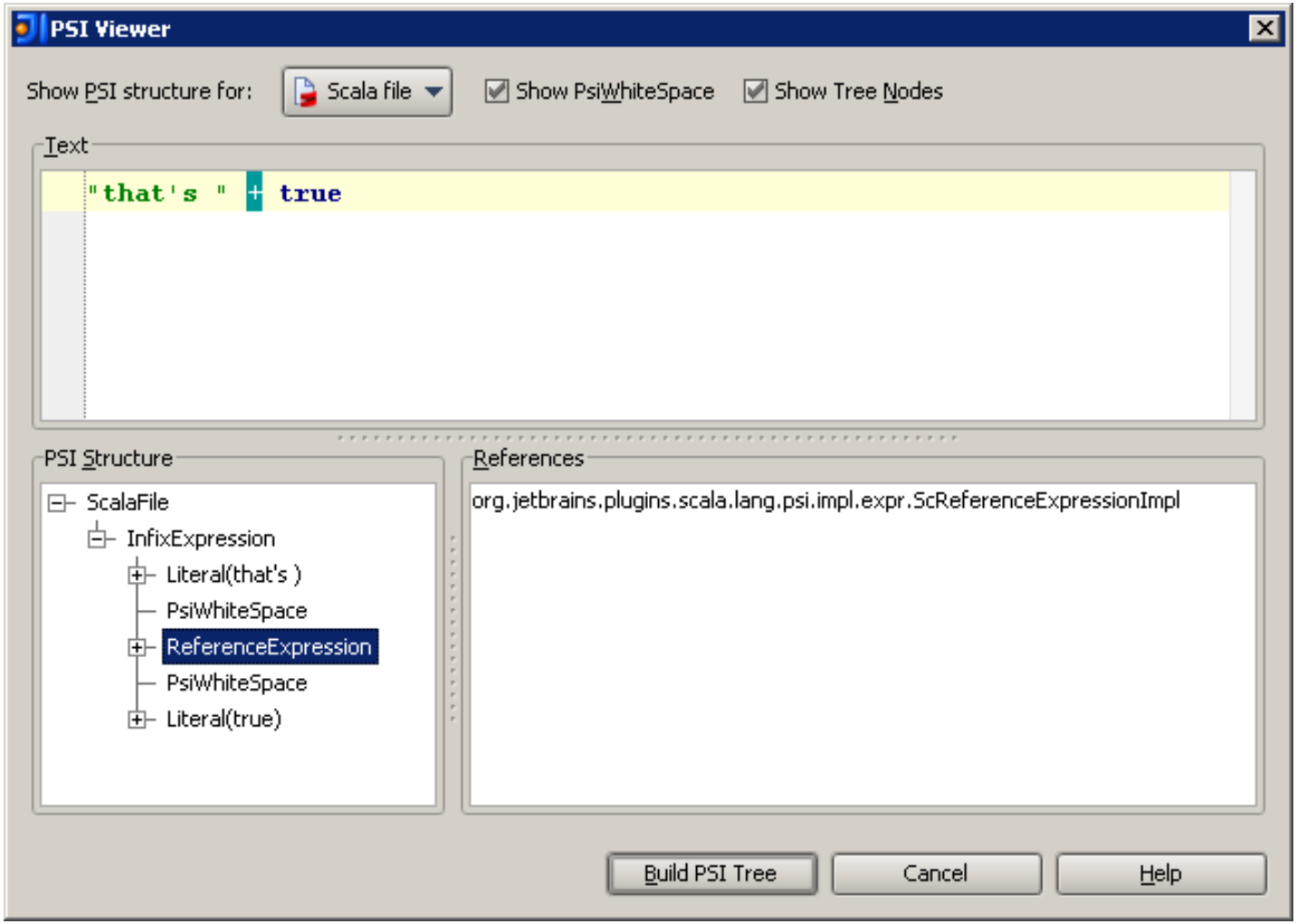
Processing example



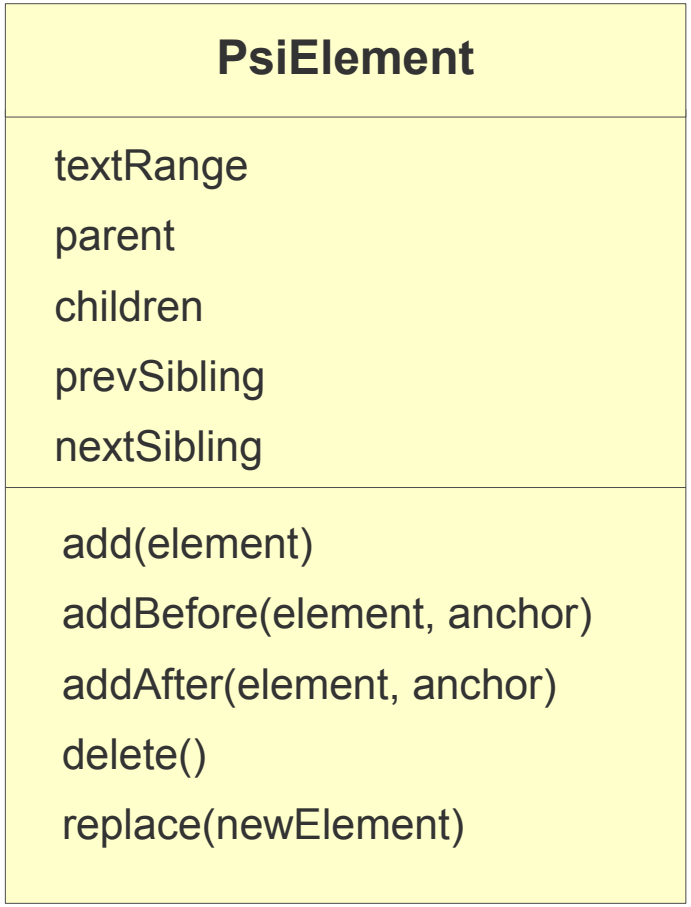
Processing example



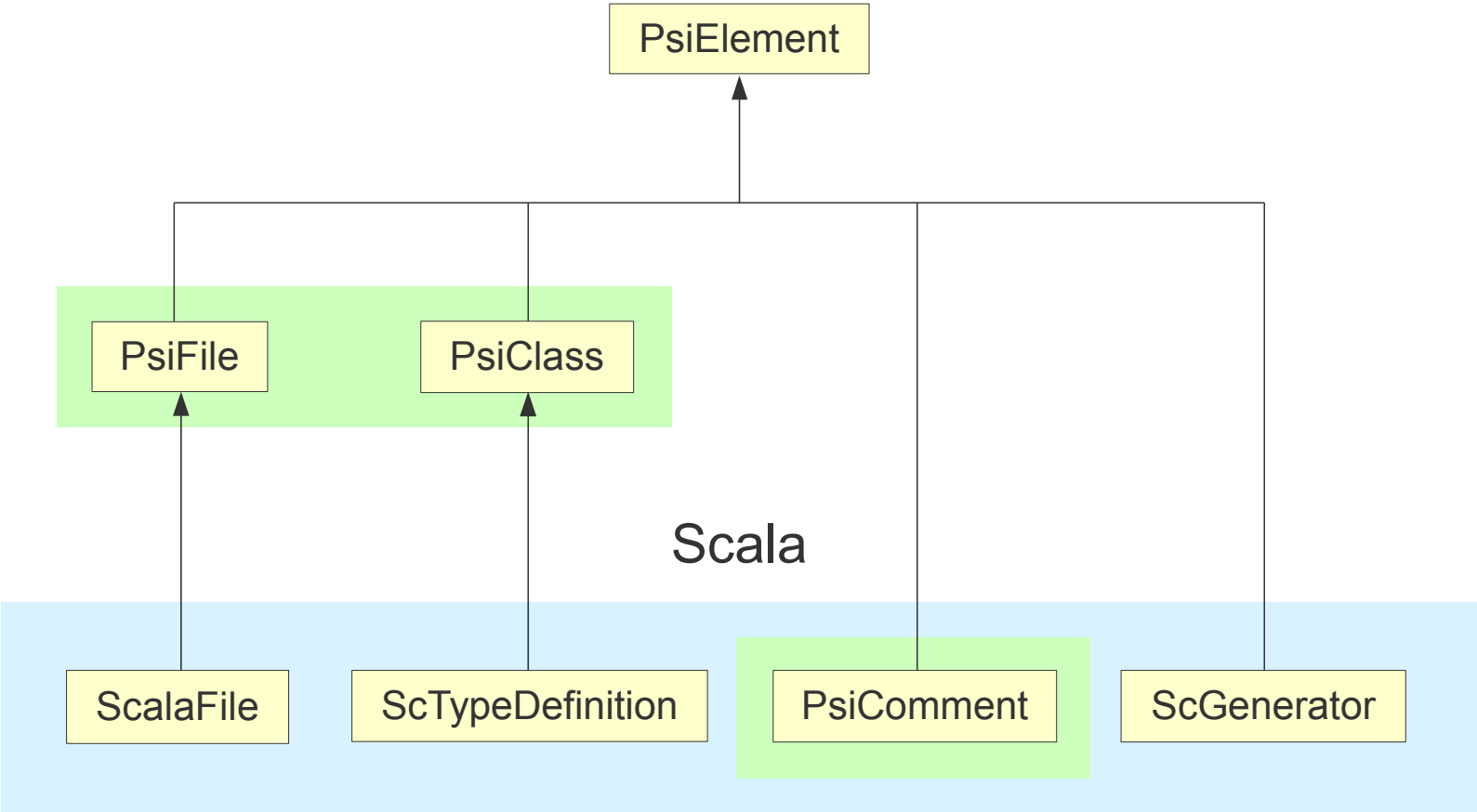
PSI viewer


PSI Viewer
Show PSI structure for:
Scala file
Show PsiWhiteSpace
Show Tree Nodes
Text
"that's " + true
PSI Structure
ScalaFile
InfixExpression
Literal(that's)
PsiWhiteSpace
ReferenceExpression
PsiWhiteSpace
Literal(true)
References
org.jetbrains.plugins.scala.lang.psi.impl.expr.ScReferenceExpressionImpl
Build PSI Tree
Cancel
Help

PSI foundation



PSI elements hierarchy



Scala Language Specification

Scala Language Specification

```
InfixExpr          ::=  PrefixExpr
                     |  InfixExpr id [nl] InfixExpr
PrefixExpr         ::=  [‘-’ | ‘+’ | ‘~’ | ‘!’] SimpleExpr
SimpleExpr         ::=  ‘new’ (ClassTemplate | TemplateBody)
                     |  BlockExpr
                     |  SimpleExpr1 [‘_’]
SimpleExpr1        ::=  Literal
                     |  Path
```

Scala Language Specification

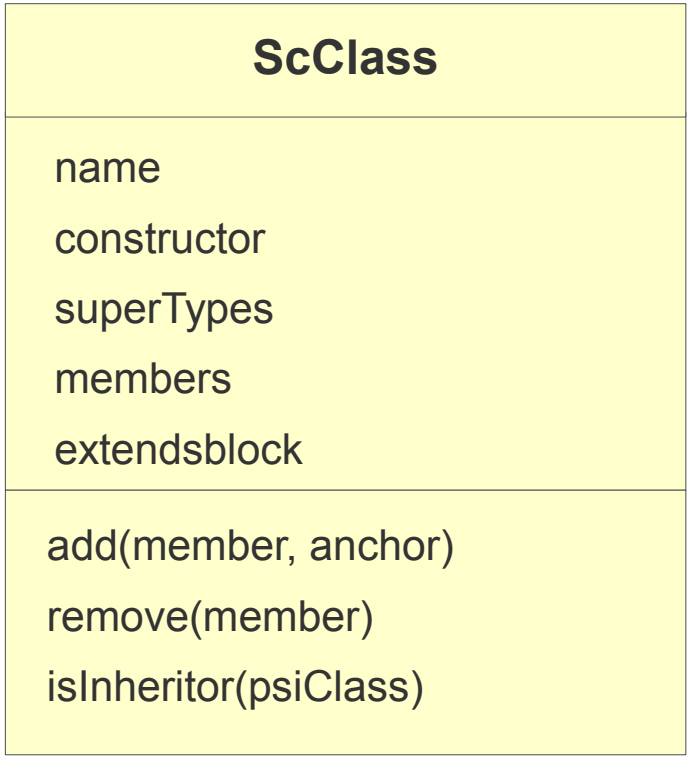
```
InfixExpr        ::=  PrefixExpr
                   |  InfixExpr id [nl] InfixExpr
PrefixExpr       ::=  ['-' | '+' | '~' | '!'] SimpleExpr
SimpleExpr       ::=  'new' (ClassTemplate | TemplateBody)
                   |  BlockExpr
                   |  SimpleExpr1 ['_']
SimpleExpr1      ::=  Literal
                   |  Path
```

6.12.3 Infix Operations

An infix operator can be an arbitrary identifier. Infix operators have precedence and associativity defined as follows:

The *precedence* of an infix operator is determined by the operator's first character. Characters are listed below in increasing order of precedence, with characters on the same line having the same precedence.

Scala domain elements



TODO

TODO

Tasks

Tasks

- Bug fixes

Tasks

- Bug fixes
- Inspections

Tasks

- Bug fixes
- Inspections
- Refactorings

Tasks

- Bug fixes
- Inspections
- Refactorings
- Intentions

Tasks

- Bug fixes
- Inspections
- Refactorings
- Intentions
- Frameworks support

Tasks

- Bug fixes
- Inspections
- Refactorings
- Intentions
- Frameworks support
- Type system improvement

Tasks

- Bug fixes
- Inspections
- Refactorings
- Intentions
- Frameworks support
- Type system improvement
- [Your most wanted feature here]

Everyone is Welcome!