

Scala, Your Next Programming Language

(or if it is good enough for Twitter, it is good enough for me)

WORLD COMP 2011

By

Dr. Mark C. Lewis
Trinity University



TRINITY
UNIVERSITY



Compiled Links

- <http://www.cs.trinity.edu/~mlewis/ScalaLinks.html>
- This is a collection of links for those who want more information on Scala.

The screenshot shows an IDE with several tabs open: Student.scala, DrawingMain.scala, DrawLeaf.scala, DrawTransform.scala, and DrawRectangle.scala. The main editor displays the following Scala code for the `DrawTransform` class:

```
package scalabook.drawing

import java.awt.Graphics2D
import javax.swing.tree.TreeNode
import scala.collection.mutable
import java.awt.geom.AffineTransform

/**
 * This class represents a transformation with a type and a value.
 * @param type the type of transformation
 * @param value the value of the transformation
 */
class DrawTransform(parent: DrawTransform) extends Drawable {
  private val subnodes = mutable.ArrayBuffer[DrawTransform]()

  private object TransformType extends Enumeration {
    val Translate, Rotate, Scale, Shear = Value
  }
  import TransformType._

  private var transformType = Translate
  private val transformValue = Array.fill(3)(0.0)

  /**
   * This method applies a transform and draws all the subnodes.
   * @param g the Graphics2D object to draw to.
   */
  def draw(g: Graphics2D) {
    val oldTrans = g.getTransform
    g.transform(buildTransform)
    subnodes.foreach(_ draw(g))
    g.setTransform(oldTrans)
  }

  private def buildTransform: AffineTransform = transformType match {
    case Translate => AffineTransform.getTranslateInstance(transformValue(0), transformValue(1))
    case Rotate => AffineTransform.getRotateInstance(transformValue(0), transformValue(1), transformValue(2))
    case Scale => AffineTransform.getScaleInstance(transformValue(0), transformValue(1))
    case Shear => AffineTransform.getShearInstance(transformValue(0), transformValue(1))
  }
}
```

The IDE also shows a Package Explorer on the left with a project named 'ScalaBook' and an Outline on the right showing the class structure. The status bar at the bottom indicates 'Writable', 'Smart Insert', and the time '16:52'.

How I got into Scala

- Grad schools and type systems
- Functional Programming and ML
- Interest in X10 and Fortress

The screenshot shows an IDE with several Scala files open. The main editor displays the following code:

```
package scalabook.drawing

import java.awt.Graphics2D
import java.awt.geom.AffineTransform
import swing._
import event._

/**
 * @param parent the DrawTransform that is the parent of this Drawable.
 */
class DrawTransform(parent: DrawTransform) extends Drawable {
  private val subnodes = mutable.Buffer[Drawable]()
  private val propPanel = new PropertyPanel()

  private object TransformType extends Enumeration {
    val Translate, Rotate, Scale, Shear = Value
  }
  import TransformType._

  private var transformType = Translate
  private val transformValue = Array.fill(3)(0.0)

  /**
   * This method applies a transform and draws all the subnodes.
   * @param g the Graphics2D object to draw to.
   */
  def draw(g: Graphics2D) {
    val oldTrans = g.getTransform
    g.transform(buildTransform)
    subnodes.foreach(_.draw(g))
    g.setTransform(oldTrans)
  }

  private def buildTransform: AffineTransform = transformType match {
    case Translate => AffineTransform.getTranslateInstance(transformValue(0), transformValue(1))
    case Rotate => AffineTransform.getRotateInstance(transformValue(0), transformValue(1), transformValue(2))
    case Scale => AffineTransform.getScaleInstance(transformValue(0), transformValue(1))
    case Shear => AffineTransform.getShearInstance(transformValue(0), transformValue(1))
  }
}
```

The IDE interface includes a Package Explorer on the left, an Outline on the right, and a console at the bottom showing "No consoles to display at this time." The status bar at the bottom indicates "Writable", "Smart Insert", and the time "16:52".

Basics of Scala

• “Scalable Language”

• Multi-Paradigm

- Productivity of scripting languages

- Expressivity of functional languages

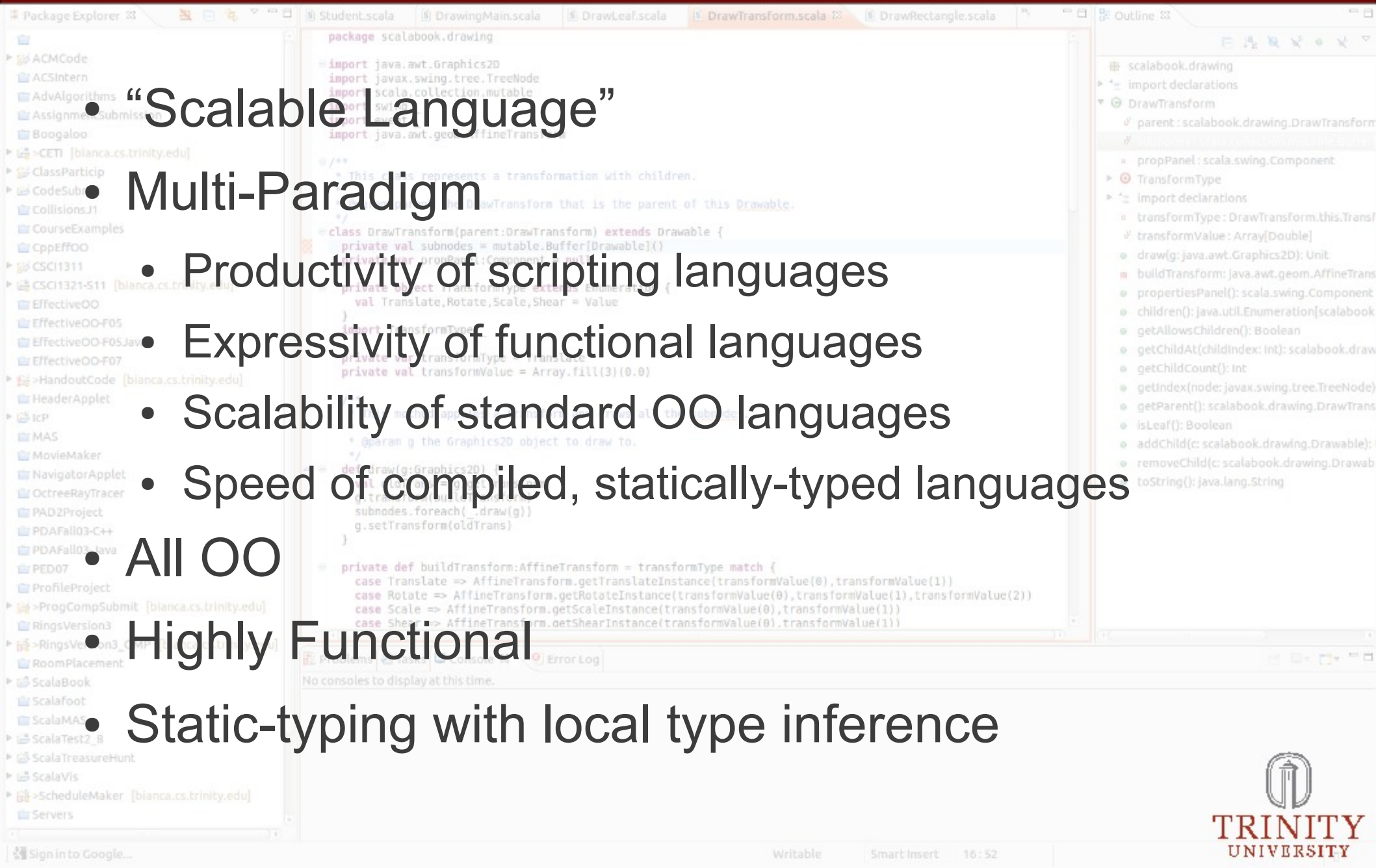
- Scalability of standard OO languages

- Speed of compiled, statically-typed languages

• All OO

• Highly Functional

• Static-typing with local type inference



The screenshot shows an IDE with several Scala files open. The main editor displays the following code:

```
package scalabook.drawing

import java.awt.Graphics2D
import javax.swing.tree.TreeNode
import scala.collection.mutable
import scala.swing.Component
import java.awt.geom.AffineTransform

/**
 * This class represents a transformation with children.
 * It is a DrawableTransform that is the parent of this Drawable.
 */
class DrawTransform(parent: DrawTransform) extends Drawable {
  private val subnodes = mutable.Buffer[Drawable]()
  private var propPanel: Component = null
  private object TransformType extends Enumeration {
    val Translate, Rotate, Scale, Shear = Value
  }
  import TransformType
  private var transformType: TransformType = Translate
  private val transformValue = Array.fill(3)(0.0)

  @param g The Graphics2D object to draw to.
  def draw(g: Graphics2D): Unit = {
    transformType match {
      case Translate => g.translate(transformValue(0), transformValue(1))
      case Rotate => g.rotate(transformValue(0), transformValue(1), transformValue(2))
      case Scale => g.scale(transformValue(0), transformValue(1))
      case Shear => g.shear(transformValue(0), transformValue(1))
    }
    subnodes.foreach(_ draw(g))
    g.setTransform(oldTrans)
  }

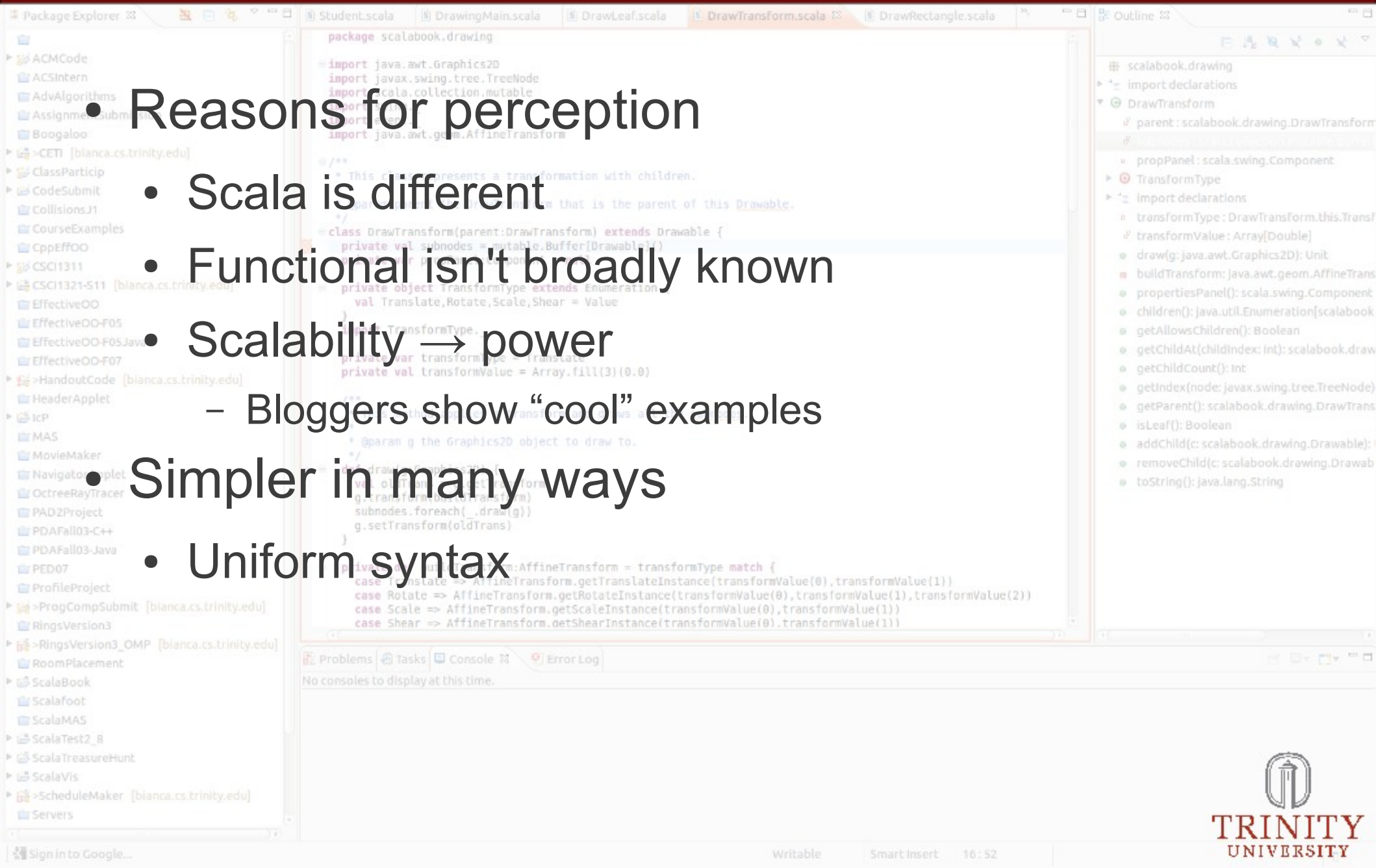
  private def buildTransform: AffineTransform = transformType match {
    case Translate => AffineTransform.getTranslateInstance(transformValue(0), transformValue(1))
    case Rotate => AffineTransform.getRotateInstance(transformValue(0), transformValue(1), transformValue(2))
    case Scale => AffineTransform.getScaleInstance(transformValue(0), transformValue(1))
    case Shear => AffineTransform.getShearInstance(transformValue(0), transformValue(1))
  }
}
```

The IDE interface includes a Package Explorer on the left, an Outline on the right, and a console at the bottom.

Too Complex?

Reasons for perception

- Scala is different
- Functional isn't broadly known
- Scalability → power
 - Bloggers show “cool” examples
- Simpler in many ways
 - Uniform syntax



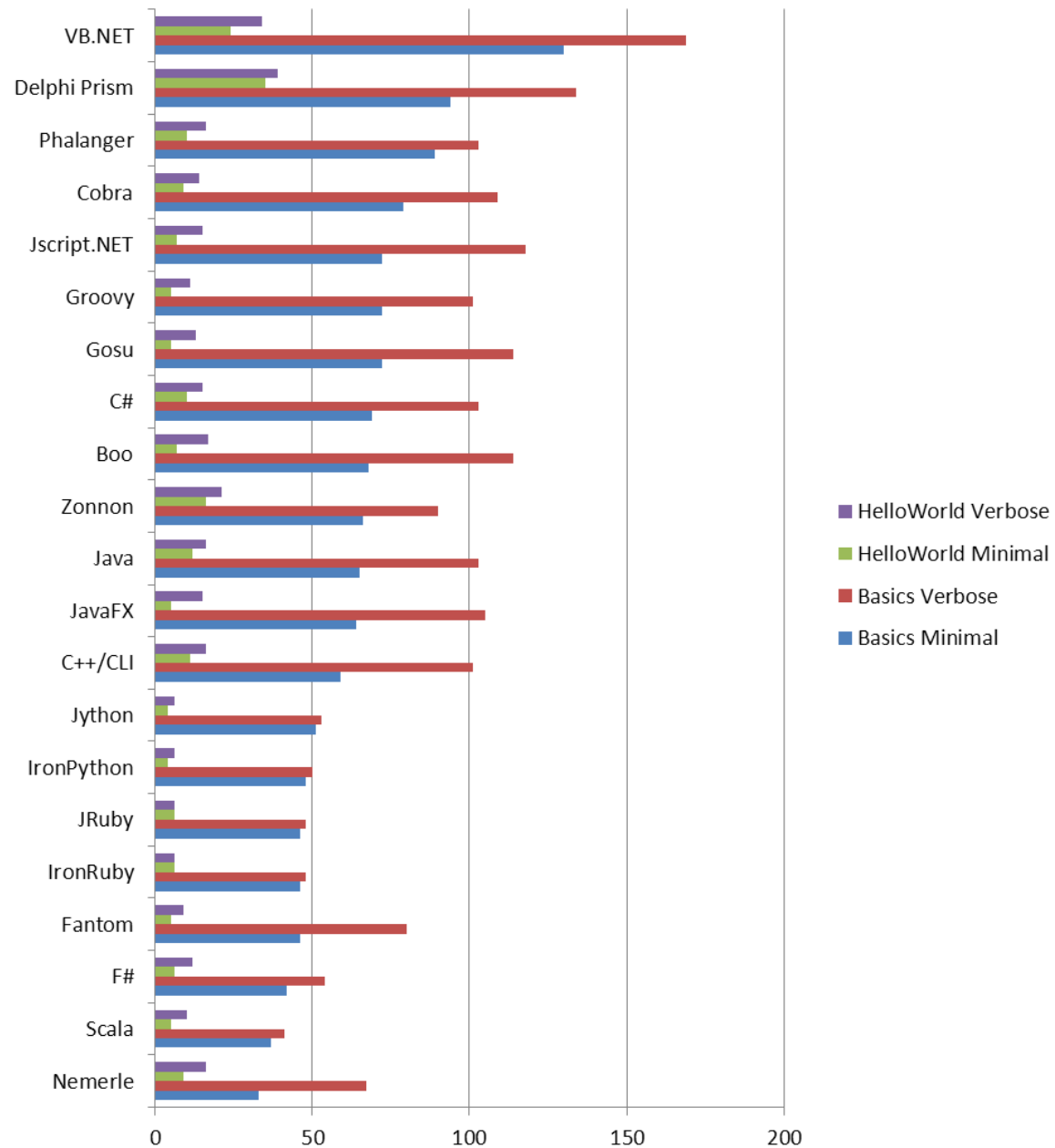
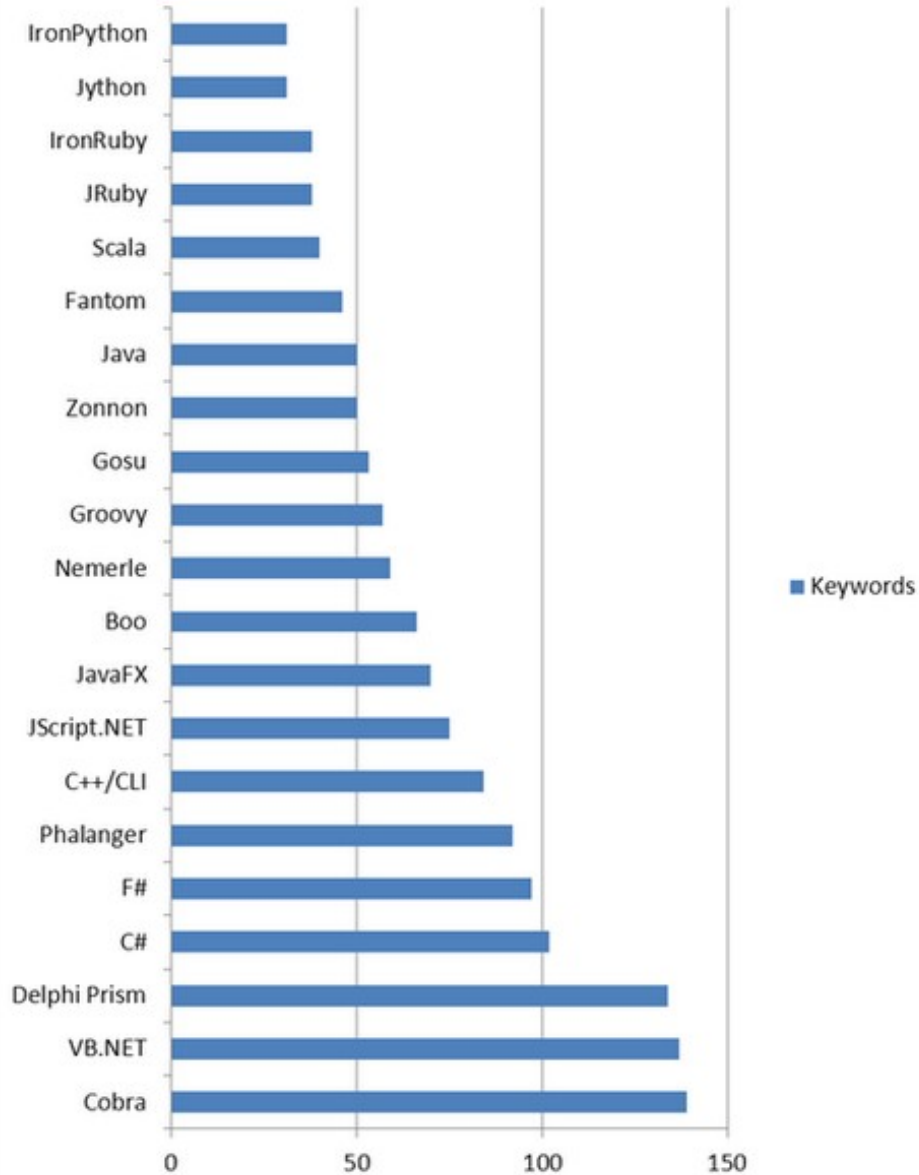
Shorter Language Specification

The background image shows an IDE window with several Scala files open: Student.scala, DrawingMain.scala, DrawLeaf.scala, DrawTransform.scala, and DrawRectangle.scala. The code in the active window includes package declarations and imports for Graphics2D, Swing, and Scala collections. A table is overlaid on the IDE, comparing the specification lengths of various programming languages. The table has four columns: 'Static Typed Language', 'Spec Length (pages)', 'Dynamic Typed Language', and 'Spec Length (pages)'. The rows list languages like C, C++, C#, Java, VB.NET, Scala, F#, Common LISP, Ruby, PHP, JavaScript, and Python 3.1. The table is styled with blue headers and alternating light blue and white rows.

Static Typed Language	Spec Length (pages)	Dynamic Typed Language	Spec Length (pages)
C	552	Common LISP	1153
C++	1325	Ruby	341
C#	553	PHP	244
Java	684	JavaScript	252
VB.NET	597	Python 3.1	119
Scala	191	Scheme	50
F#	250		

Fewer Keywords

Keywords



Past Complaints

• Documentation

- API
- Books
- Web resources

• Tools

- Improved IDE support

• Support

- Typesafe

The screenshot shows an IDE with several Scala files open. The main editor displays the following code:

```
package scalabook.drawing

import java.awt.Graphics2D
import javax.swing.tree.TreeNode
import scala.collection.mutable
import java.awt.geom.AffineTransform

/**
 * This class represents a transformation with children.
 *
 * @param parent the DrawTransform that is the parent of this Drawable.
 */
class DrawTransform(parent: DrawTransform) extends Drawable {
  private val subnodes = mutable.Buffer[Drawable]()
  private var propPanel: Component = null

  private object TransformType extends Enumeration {
    val Translate, Rotate, Scale, Shear = Value
  }
  import TransformType

  private var transformType = Translate
  private val transformValue = Array.fill(3)(0.0)

  /**
   * This method applies a transform and draws all the subnodes.
   *
   * @param g the Graphics2D object to draw to.
   */
  def draw(g: Graphics2D) {
    val oldTransform = g.getTransform
    val inst = (transformType match {
      case Translate => AffineTransform.getTranslateInstance(transformValue(0), transformValue(1))
      case Rotate => AffineTransform.getRotateInstance(transformValue(0), transformValue(1), transformValue(2))
      case Scale => AffineTransform.getScaleInstance(transformValue(0), transformValue(1))
      case Shear => AffineTransform.getShearInstance(transformValue(0), transformValue(1))
    })
    g.setTransform(oldTransform)
  }

  private def buildTransform: AffineTransform = transformType match {
    case Translate => AffineTransform.getTranslateInstance(transformValue(0), transformValue(1))
    case Rotate => AffineTransform.getRotateInstance(transformValue(0), transformValue(1), transformValue(2))
    case Scale => AffineTransform.getScaleInstance(transformValue(0), transformValue(1))
    case Shear => AffineTransform.getShearInstance(transformValue(0), transformValue(1))
  }
}
```

The IDE interface includes a Package Explorer on the left, an Outline on the right, and a Console at the bottom. The status bar at the bottom shows 'Writable', 'Smart Insert', and '16:52'.

Current Challenges

• Compile Time

- Smart, multipass compiler
- Optimization in progress

• Binary Compatibility

- Changes in traits require recompilation

• Uniform Style

- Many paradigms
- Need guidelines
- Tools coming

The screenshot shows an IDE with several Scala files open. The main editor displays the `scalabook.drawing` package with the `DrawTransform` class. The code includes imports for `java.awt.Graphics2D`, `javax.swing.tree.TreeNode`, `scala.collection.mutable`, and `java.awt.geom.AffineTransform`. The `DrawTransform` class extends `Drawable` and has a `draw` method that takes a `Graphics2D` object and a `TreeNode` object. It also has a `buildTransform` method that returns an `AffineTransform` object based on the `TransformType` and `transformValue` array.

```
package scalabook.drawing

import java.awt.Graphics2D
import javax.swing.tree.TreeNode
import scala.collection.mutable
import java.awt.geom.AffineTransform

/**
 * This class represents a transformation with children
 */
class DrawTransform(parent: DrawTransform) extends Drawable {
  private val subnode = mutable.Buffer[Drawable]()
  private object TransformType extends Enumeration {
    val Translate, Rotate, Scale, Shear = Value
  }
  import TransformType
  private val transformValue = Array.fill(3)(0.0)

  /**
   * This method traverses the transform down all the children
   * regarding the wrapped object to draw.
   */
  def draw(g: Graphics2D) {
    val oldTrans = g.getTransform
    subnode.foreach { child => child.draw(g) }
    g.setTransform(oldTrans)
  }

  private def buildTransform: AffineTransform = transformType match {
    case Translate => AffineTransform.getTranslateInstance(transformValue(0), transformValue(1))
    case Rotate => AffineTransform.getRotateInstance(transformValue(0), transformValue(1), transformValue(2))
    case Scale => AffineTransform.getScaleInstance(transformValue(0), transformValue(1))
    case Shear => AffineTransform.getShearInstance(transformValue(0), transformValue(1))
  }
}
```

Jobs



Language Intro

- General rules
 - All values are objects
 - Operators are methods
 - All expressions are valid statements
 - Almost everything is an expression
- Declarations
 - Begin with keyword
 - val, var, def, type, class, object, trait
- Value parameters use ()
- Type parameters use []

The screenshot shows an IDE with several Scala files open. The main file, `DrawTransform.scala`, contains the following code:

```
package scalabook.drawing

import java.awt.Graphics2D
import javax.swing.tree.TreeNode
import java.util.Collection.mutable
import swing._
import event._
import java.awt.geom.AffineTransform

/**
 * This class represents a transformation with children.
 * @param parent the DrawTransform that is the parent of this Drawable.
 */
abstract class DrawTransform(parent: DrawTransform) extends Drawable {
  private val subnodes = mutable.Buffer[Drawable]()
  private var propPanel: Component = null

  import TransformType._

  private val transformValue = Array(1.0, 0.0, 0.0, 1.0)

  /**
   * This method applies a transform and draws all the subnodes.
   */
  def draw(g: Graphics2D): Unit = {
    val oldTrans = g.getTransform
    g.setTransform(transformValue)
    subnodes.foreach(_.draw(g))
    g.setTransform(oldTrans)
  }

  case Rotate => AffineTransform.getRotateInstance(transformValue(0), transformValue(1), transformValue(2))
  case Scale => AffineTransform.getScaleInstance(transformValue(0), transformValue(1))
  case Translate => AffineTransform.getTranslateInstance(transformValue(0), transformValue(1))
}

class DrawTransform(parent: DrawTransform) {
  // ...
}
```

Syntactic Sugar

• Semicolon inference

- Newlines become semicolons when it fits
- Explicit works too

• Operator notation

- Leave off dot and parens
- Methods with arity 0 or 1

```
package scalabook.drawing

import java.awt.Graphics2D
import javax.swing.tree.TreeNode
import scala.collection.mutable
import java.awt.geom.AffineTransform

/**
 * This class represents a transformation with children.
 */
class DrawTransform(parent: DrawTransform) extends Drawable {
  private val subnodes = mutable.Buffer[Drawable]()
  private val transformType: TransformType = TransformType.Translate
  private object TransformType extends Enumeration {
    val Translate, Rotate, Scale, Shear = Value
  }
  import TransformType
  private val transformValue = Array.fill(3)(0.0)

  /**
   * This method applies a transform and draws all the subnodes.
   */
  def draw(g: Graphics2D) {
    val oldTransform = g.getTransform()
    g.setTransform(transformValue)
    subnodes.foreach(_.draw(g))
    g.setTransform(oldTransform)
  }

  private def buildTransform: AffineTransform = transformType match {
    case Translate => AffineTransform.getTranslateInstance(transformValue(0), transformValue(1))
    case Rotate => AffineTransform.getRotateInstance(transformValue(0), transformValue(1), transformValue(2))
    case Scale => AffineTransform.getScaleInstance(transformValue(0), transformValue(1))
    case Shear => AffineTransform.getShearInstance(transformValue(0), transformValue(1))
  }
}
```

Outline:

- scalabook.drawing
 - import declarations
 - DrawTransform
 - parent: scalabook.drawing.DrawTransform
 - propPanel: scala.swing.Component
 - TransformType
 - import declarations
 - transformType: DrawTransform.this.TransformType
 - transformValue: Array[Double]
 - draw(g: java.awt.Graphics2D): Unit
 - buildTransform: java.awt.geom.AffineTransform
 - propertiesPanel(): scala.swing.Component
 - children(): java.util.Enumeration[scalabook.drawing.Drawable]
 - getAllowsChildren(): Boolean
 - getChildAt(childIndex: Int): scalabook.drawing.Drawable
 - getChildCount(): Int
 - getIndex(node: javax.swing.tree.TreeNode): Int
 - getParent(): scalabook.drawing.DrawTransform
 - isLeaf(): Boolean
 - addChild(c: scalabook.drawing.Drawable): Unit
 - removeChild(c: scalabook.drawing.Drawable): Unit
 - toString(): java.lang.String

Usage Methods

- REPL for simple tests
- Scripting for short programs
 - Good for gluing things together
- Applications
 - Declare object with main
 - Model is much like Java

The screenshot shows an IDE with several tabs open: Student.scala, DrawingMain.scala, DrawLeaf.scala, DrawTransform.scala, and DrawRectangle.scala. The main editor displays the following Scala code:

```
package scalabook.drawing

import java.awt.Graphics2D
import java.awt.geom.AffineTransform
import swing._
import event._
import java.awt.geom.AffineTransform

/**
 * This class represents a drawable object that can be drawn to.
 * @param parent the DrawTransform that is the parent of this Drawable.
 */
class DrawTransform(parent: DrawTransform) extends Drawable {
  private val transformType = TransformType._
  private val transformValue = Array.fill(3)(0.0)

  /**
   * Draw this object to the given Graphics2D object.
   */
  def draw(g: Graphics2D) {
    val oldTrans = g.getTransform()
    transformType match {
      case Translate => AffineTransform.getTranslateInstance(transformValue(0), transformValue(1))
      case Rotate => AffineTransform.getRotateInstance(transformValue(0), transformValue(1), transformValue(2))
      case Scale => AffineTransform.getScaleInstance(transformValue(0), transformValue(1))
      case Shear => AffineTransform.getShearInstance(transformValue(0), transformValue(1))
    }
    g.setTransform(oldTrans)
  }

  private def buildTransform: AffineTransform = transformType match {
    case Translate => AffineTransform.getTranslateInstance(transformValue(0), transformValue(1))
    case Rotate => AffineTransform.getRotateInstance(transformValue(0), transformValue(1), transformValue(2))
    case Scale => AffineTransform.getScaleInstance(transformValue(0), transformValue(1))
    case Shear => AffineTransform.getShearInstance(transformValue(0), transformValue(1))
  }
}
```

The IDE also shows a Package Explorer on the left with a project named 'ScalaBook' and an Outline on the right showing the class hierarchy for 'scalabook.drawing.DrawTransform'.

Difference for Classes

- Take arguments
 - No body required
 - Code in body run at construction
- Special methods
 - Symbols
 - Property assignment
 - apply
 - update
- Case classes

```
package scalabook.drawing

import java.awt.Graphics2D
import java.awt.geom.AffineTransform
import java.awt.geom.Transform

import swing._
import event._

import java.awt.geom.AffineTransform

class DrawTransform(parent: DrawTransform, transformType: TransformType) extends Drawable {
  private var propPanel: Component = null

  private object TransformType extends Enumeration {
    val Translate, Rotate, Scale, Shear = Value
  }

  private var transformType = Translate
  private val transformValue = Array.fill(3)(0.0)

  def draw(g: Graphics2D): Unit = {
    this method applies a transform and draws all the subnodes.
    g.translate(transformValue(0), transformValue(1))
    g.rotate(transformValue(2))
    g.scale(transformValue(3), transformValue(4))
    g.shear(transformValue(5), transformValue(6))
    subnodes.foreach(_ draw(g))
    g.setTransform(oldTrans)
  }

  private def buildTransform: AffineTransform = transformType match {
    case Translate => AffineTransform.getTranslateInstance(transformValue(0), transformValue(1))
    case Rotate => AffineTransform.getRotateInstance(transformValue(0), transformValue(1), transformValue(2))
    case Scale => AffineTransform.getScaleInstance(transformValue(3), transformValue(4))
    case Shear => AffineTransform.getShearInstance(transformValue(5), transformValue(6))
  }
}
```

Outline:

- scalabook.drawing
 - import declarations
 - DrawTransform
 - parent: scalabook.drawing.DrawTransform
 - propPanel: scala.swing.Component
 - TransformType
 - import declarations
 - transformType: DrawTransform.this.TransformType
 - transformValue: Array[Double]
 - draw(g: java.awt.Graphics2D): Unit
 - buildTransform: java.awt.geom.AffineTransform
 - propertiesPanel(): scala.swing.Component
 - children(): java.util.Enumeration[scalabook.drawing.Drawable]
 - getAllowsChildren(): Boolean
 - getChildAt(childIndex: Int): scalabook.drawing.Drawable
 - getChildCount(): Int
 - getIndex(node: javax.swing.tree.TreeNode): Int
 - getParent(): scalabook.drawing.DrawTransform
 - isLeaf(): Boolean
 - addChild(c: scalabook.drawing.Drawable): Unit
 - removeChild(c: scalabook.drawing.Drawable): Unit
 - toString(): java.lang.String

Writable Smart Insert 16:52

Object Declarations

- Creates singleton objects
- No static in Scala
- Companion objects
- Apply method commonly used for object construction
- No arguments
- Can inherit

The screenshot shows an IDE with several Scala files open. The main editor displays the following code:

```
package scalabook.drawing

import java.awt.Graphics2D
import java.awt.geom.AffineTransform
import java.awt.geom.Transform
import java.awt.geom.TransformType
import event._

class Drawable {
  // ...
}

class DrawTransform(parent: DrawTransform) extends Drawable {
  private val subnodes = mutable.Buffer[Drawable]()

  // ...
}

private object TransformType extends Enumeration {
  val Translate, Rotate, Scale, Shear = Value
}

import TransformType

private val transformValue = Array.fill(3)(0.0)

// ...
}

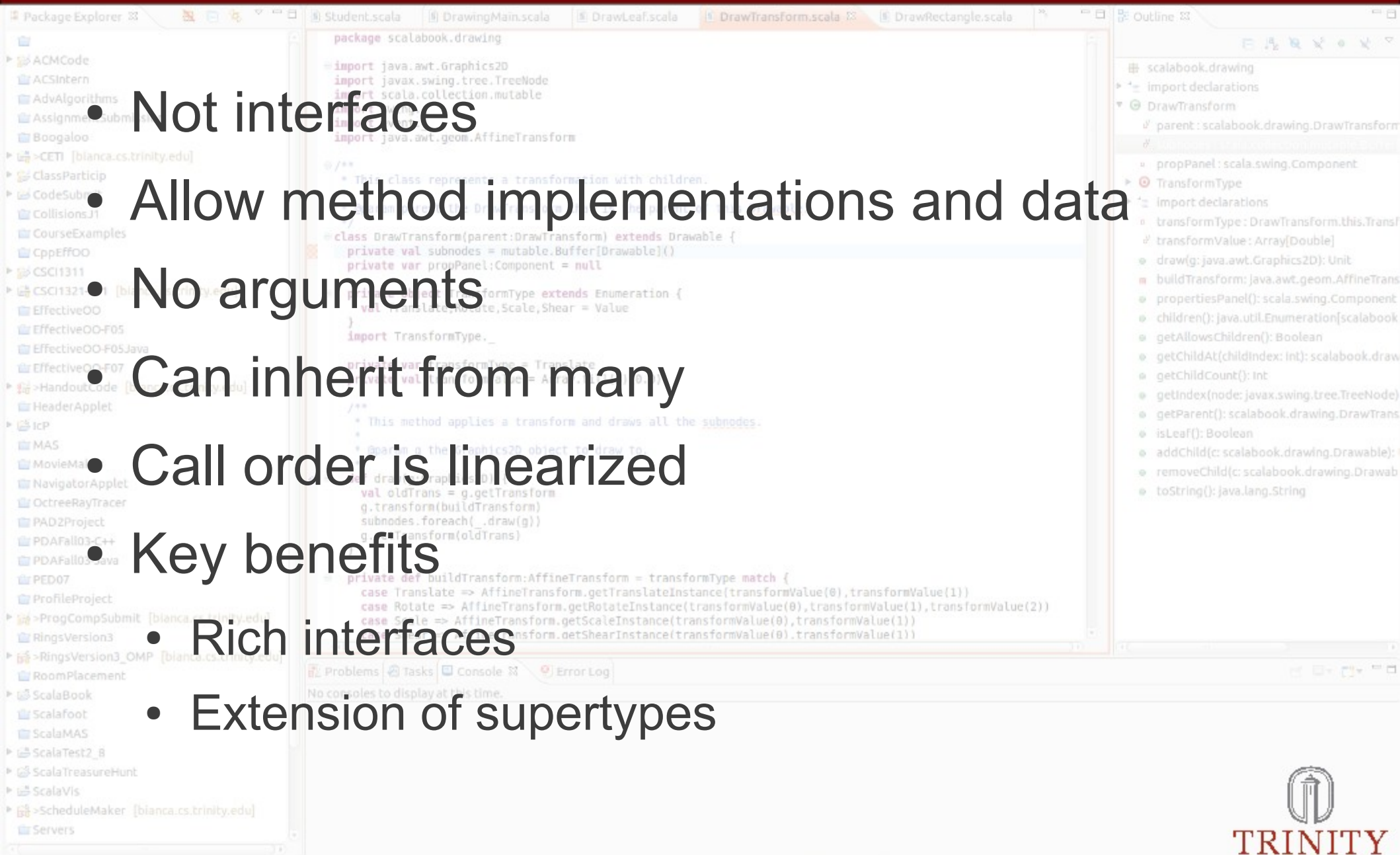
def buildTransform: AffineTransform = transformType match {
  case Translate => AffineTransform.getTranslateInstance(transformValue(0), transformValue(1))
  case Rotate => AffineTransform.getRotateInstance(transformValue(0), transformValue(1), transformValue(2))
  case Scale => AffineTransform.getScaleInstance(transformValue(0), transformValue(1))
  case Shear => AffineTransform.getShearInstance(transformValue(0), transformValue(1))
}
```

The Outline view on the right shows the following structure:

- scalabook.drawing
 - import declarations
 - DrawTransform
 - parent: scalabook.drawing.DrawTransform
 - propPanel: scala.swing.Component
 - TransformType
 - import declarations
 - transformType: DrawTransform.this.TransformType
 - transformValue: Array[Double]
 - draw(g): java.awt.Graphics2D: Unit
 - buildTransform: java.awt.geom.AffineTransform: AffineTransform
 - propertiesPanel(): scala.swing.Component
 - children(): java.util.Enumeration[scalabook.drawing.Drawable]: Enumeration
 - getAllowsChildren(): Boolean
 - getChildAt(childIndex: Int): scalabook.drawing.Drawable
 - getChildCount(): Int
 - getIndex(node: javax.swing.tree.TreeNode): Int
 - getParent(): scalabook.drawing.DrawTransform
 - isLeaf(): Boolean
 - addChild(c: scalabook.drawing.Drawable): Unit
 - removeChild(c: scalabook.drawing.Drawable): Unit
 - toString(): java.lang.String

Traits

- Not interfaces
- Allow method implementations and data
- No arguments
- Can inherit from many
- Call order is linearized
- Key benefits
 - Rich interfaces
 - Extension of supertypes



Standard

The image shows a screenshot of an IDE with Scala code and a class hierarchy diagram overlaid. The code is in the file `DrawTransform.scala` within the `scalabook.drawing` package. It defines the `DrawTransform` class, which extends `Drawable` and implements `Iterable`. The class has a `parent` of type `DrawTransform` and a `propPanel` of type `Component`. It also has an enumeration `TransformType` with values `Translate`, `Rotate`, `Scale`, and `Shear`. The `draw` method applies a transformation to all subnodes. The `buildTransform` method returns an `AffineTransform` based on the `transformType`.

```
package scalabook.drawing

import java.awt.Graphics2D
import javax.swing.tree.TreeNode
import scala.collection.mutable
import swing._
import event._
import java.awt.geom.AffineTransform

/**
 * This class represents a transformation with children.
 *
 * @param parent the DrawTransform that is the parent of this Drawable.
 */
class DrawTransform(parent: DrawTransform) extends Drawable {
  private val subnodes = mutable.Buffer[Drawable]()
  private var propPanel: Component = null

  private object TransformType extends Enumeration {
    val Translate, Rotate, Scale, Shear = Value
  }
  import TransformType._

  private var transformType = Translate
  private val transformValue = Array.fill(3)(0.0)

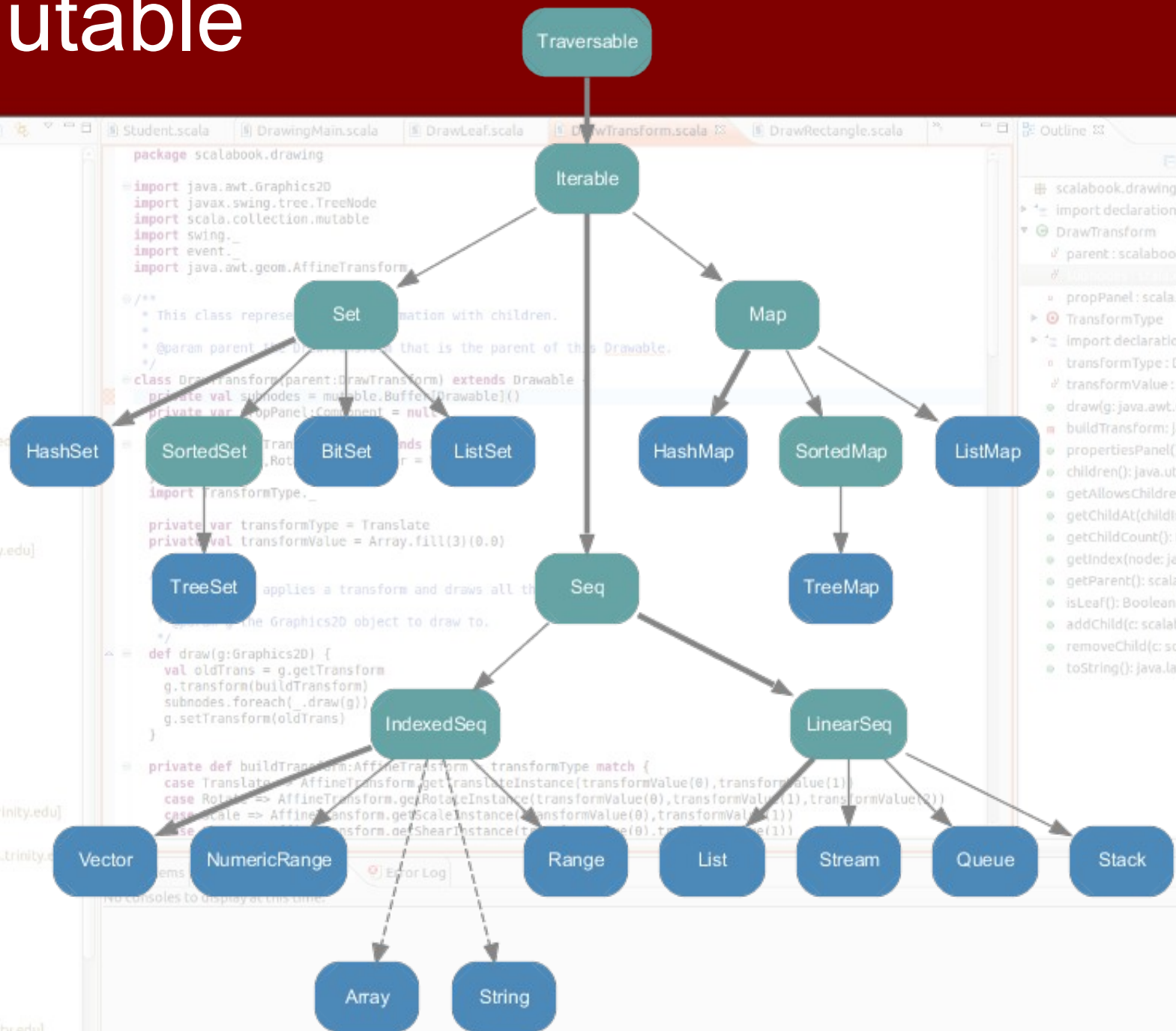
  /**
   * This method applies a transform to all the subnodes.
   *
   * @param g the Graphics2D
   */
  def draw(g: Graphics2D) {
    val oldTrans = g.getTransform()
    g.transform(buildTransform())
    subnodes.foreach(_.draw(g))
    g.setTransform(oldTrans)
  }

  private def buildTransform(): AffineTransform = transformType match {
    case Translate => AffineTransform.getInstance(transformValue(0), transformValue(1))
    case Rotate => AffineTransform.getInstance(transformValue(0), transformValue(1), transformValue(2))
    case Scale => AffineTransform.getInstance(transformValue(0), transformValue(1), transformValue(2))
    case Shear => AffineTransform.getInstance(transformValue(0), transformValue(1), transformValue(2))
  }
}
```

The class hierarchy diagram shows the following structure:

- `Traversable` (parent)
- `Iterable` (child of `Traversable`)
- `Seq` (child of `Iterable`)
- `Set` (child of `Iterable`)
- `Map` (child of `Iterable`)
- `IndexedSeq` (child of `Seq`)
- `LinearSeq` (child of `Seq`)
- `SortedSet` (child of `Set`)
- `BitSet` (child of `Set`)
- `SortedMap` (child of `Map`)

Immutable



```
package scalabook.drawing

import java.awt.Graphics2D
import javax.swing.tree.TreeNode
import scala.collection.mutable
import swing._
import event._
import java.awt.geom.AffineTransform

/**
 * This class represents a Drawable with children.
 * @param parent The parent of this Drawable.
 */
class DrawTransform(parent: DrawTransform) extends Drawable {
  private val subnodes = mutable.Buffer[Drawable]()
  private var propPanel: Component = null

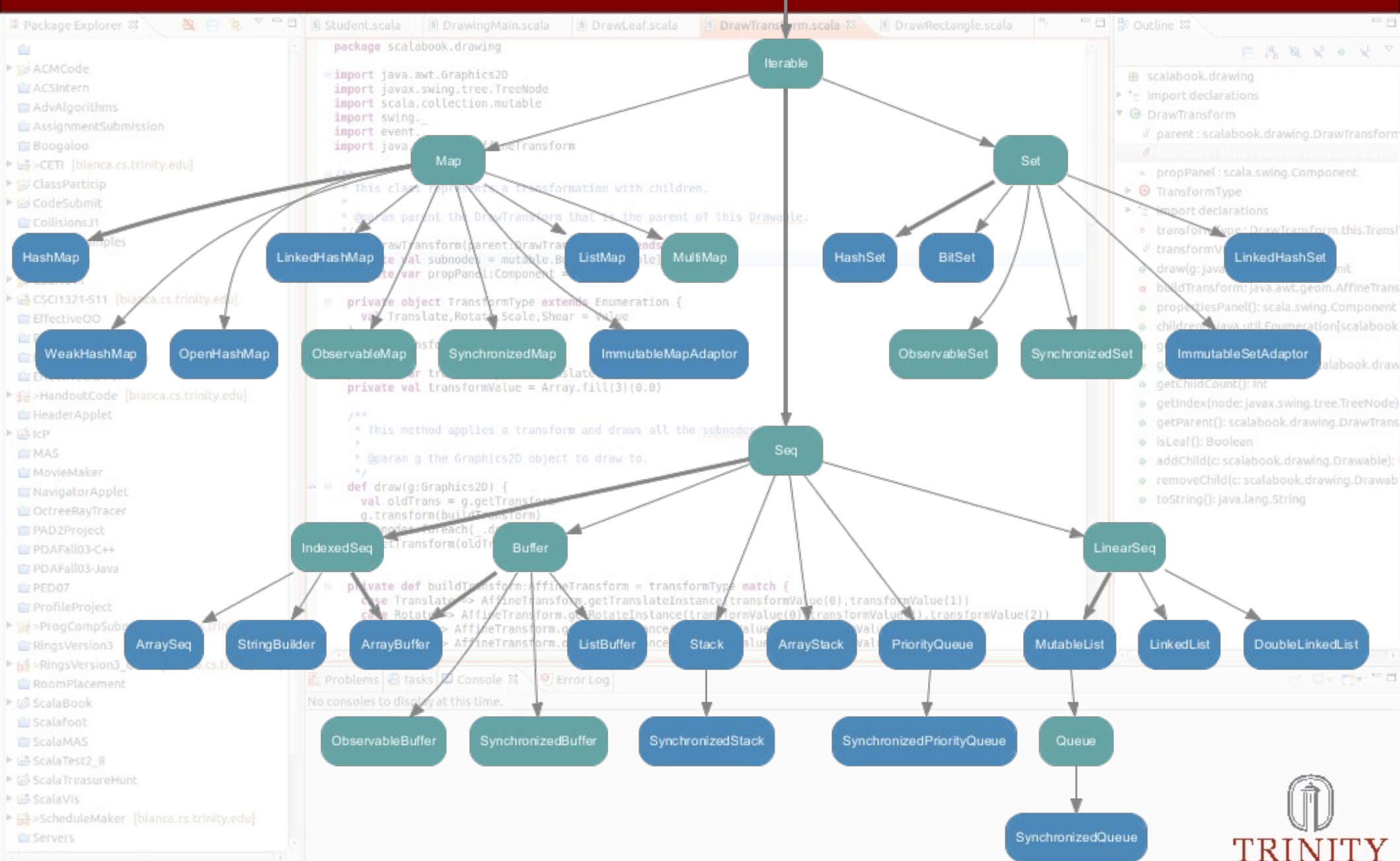
  import transformType._

  private var transformType = Translate
  private val transformValue = Array.fill(3)(0.0)

  def draw(g: Graphics2D) {
    val oldTrans = g.getTransform
    g.transform(buildTransform)
    subnodes.foreach(_ draw(g))
    g.setTransform(oldTrans)
  }

  private def buildTransform: AffineTransform = transformType match {
    case Translate => AffineTransform.getTranslateInstance(transformValue(0), transformValue(1))
    case Rotate => AffineTransform.getRotateInstance(transformValue(0), transformValue(1), transformValue(2))
    case Scale => AffineTransform.getScaleInstance(transformValue(0), transformValue(1))
    case Shear => AffineTransform.getShearInstance(transformValue(0), transformValue(1))
  }
}
```

Mutable



Match/Patterns

- Match is not just switch
- Cases can be patterns and bind variables.
- Examples of patterns
 - Tuples
 - Array, List, etc.
 - RegEx
 - XML
 - much more
- No break

The screenshot shows an IDE with several Scala files open. The main editor displays the following code:

```
package scalabook.drawing

import java.awt.Graphics2D
import java.awt.geom.AffineTransform
import swing._
import event._

import java.awt.geom.AffineTransform

/**
 * @param parent the DrawTransform that is the parent of this Drawable.
 */
class DrawTransform(parent: DrawTransform) extends Drawable {
  private def buildTransform(): AffineTransform = {
    val transformType = TransformType
    private val transformValue = Array.fill(3)(0.0)

    private def draw(g: Graphics2D) {
      val oldTrans = g.getTransform
      g.transform(buildTransform)
      subnodes.foreach(_ draw(g))
      g.setTransform(oldTrans)
    }

    private def buildTransform: AffineTransform = transformType match {
      case Translate => AffineTransform.getTranslateInstance(transformValue(0), transformValue(1))
      case Rotate => AffineTransform.getRotateInstance(transformValue(0), transformValue(1), transformValue(2))
      case Scale => AffineTransform.getScaleInstance(transformValue(0), transformValue(1))
      case Shear => AffineTransform.getShearInstance(transformValue(0), transformValue(1))
    }
  }
}
```

The Outline pane on the right shows the project structure:

- scalabook.drawing
 - import declarations
 - DrawTransform
 - parent: scalabook.drawing.DrawTransform
 - propPanel: scala.swing.Component
 - TransformType
 - import declarations
 - transformType: DrawTransform.this.TransformType
 - transformValue: Array[Double]
 - draw(g): java.awt.Graphics2D: Unit
 - buildTransform: java.awt.geom.AffineTransform
 - propertiesPanel(): scala.swing.Component
 - children(): java.util.Enumeration[scalabook.drawing.Drawable]
 - getAllowsChildren(): Boolean
 - getChildAt(childIndex: Int): scalabook.drawing.Drawable
 - getChildCount(): Int
 - getIndex(node: javax.swing.tree.TreeNode): Int
 - getParent(): scalabook.drawing.DrawTransform
 - isLeaf(): Boolean
 - addChild(c: scalabook.drawing.Drawable): Unit
 - removeChild(c: scalabook.drawing.Drawable): Unit
 - toString(): java.lang.String

For Expressions

- Not your normal for
- Technically for-each
- More options
- Multiple generators
- Variable declarations
- If guards
- Patterns
- No break, continue, or goto

```
package scalabook.drawing

import java.awt.Graphics2D
import java.awt.geom.AffineTransform
import swing._
import event._

class DrawTransform(parent: DrawTransform) extends Drawable {
  private val subnodes = mutable.Buffer[Drawable]()

  @param parent the DrawTransform that is the parent of this Drawable.
  /**
   * @param g the Graphics2D object to draw to.
   */
  def draw(g: Graphics2D) {
    oldTrans = g.getTransform
    transform(buildTransform)
    subnodes.foreach(_._draw(g))
    g.setTransform(oldTrans)
  }

  private def buildTransform: AffineTransform = transformType match {
    case Translate => AffineTransform.getTranslateInstance(transformValue(0), transformValue(1))
    case Rotate => AffineTransform.getRotateInstance(transformValue(0), transformValue(1), transformValue(2))
    case Scale => AffineTransform.getScaleInstance(transformValue(0), transformValue(1))
    case Shear => AffineTransform.getShearInstance(transformValue(0), transformValue(1))
  }
}
```


Implicit Conversions

• “Pimp my Interface”

• Strict rules

- Must be in scope
- Only one applied
- Allows extension of
 - java.lang.String
 - Arrays
 - Any code you didn't write

```
package scalabook.drawing

import java.awt.Graphics2D
import javax.swing.tree.TreeNode
import java.util.ArrayList
import java.awt.geom.AffineTransform

/**
 * This class represents a transformation with children.
 * @param parent the DrawTransform that is the parent of this Drawable.
 */
class DrawTransform(parent: DrawTransform) extends Drawable {
  private val subnodes = mutable.Buffer[Drawable]()
  private var propPanel: Component = null

  val transformType: TransformType = TransformType.Translate
  private val transformValue = Array.fill(3)(0.0)

  import TransformType

  /**
   * This method applies a transform and draws all the subnodes.
   */
  def draw(g: Graphics2D) {
    val oldTrans = g.getTransform()
    g.transform(buildTransform)
    subnodes.foreach(_.draw(g))
    g.setTransform(oldTrans)
  }

  private def buildTransform: AffineTransform = transformType match {
    case Translate => AffineTransform.getTranslateInstance(transformValue(0), transformValue(1))
    case Rotate => AffineTransform.getRotateInstance(transformValue(0), transformValue(1), transformValue(2))
    case Scale => AffineTransform.getScaleInstance(transformValue(0), transformValue(1))
    case Shear => AffineTransform.getShearInstance(transformValue(0), transformValue(1))
  }
}
```

Useful Scripting

- scala.sys.process added in 2.9
- Simple system calls
- Piping between programs
- Conditional calls
- Allows Java library calls
- Glue things together

The screenshot shows an IDE with several Scala files open. The main editor displays the following code:

```
package scalabook.drawing

import java.awt.Graphics2D
import java.awt.geom.AffineTransform
import java.awt.event._

class DrawTransform(parent: DrawTransform) extends Drawable {
  private val subnodes = mutable.Buffer[Drawable]()

  /**
   * @param parent the DrawTransform that is the parent of this Drawable.
   */
  class DrawTransform(parent: DrawTransform) extends Drawable {
    private val subnodes = mutable.Buffer[Drawable]()

    private object TransformType extends Enumeration {
      val Translate, Rotate, Scale, Shear = Value
    }

    import TransformType

    private var transformType = Translate
    private val transformValue = Array.fill(3)(0.0)

    /**
     * This method applies a transform and draws all subnodes.
     */
    def draw(g: Graphics2D) {
      val oldTrans = g.getTransform
      g.transform(buildTransform)

      subnodes.foreach(_.draw(g))
      g.setTransform(oldTrans)
    }

    private def buildTransform: AffineTransform = transformType match {
      case Translate => AffineTransform.getTranslateInstance(transformValue(0), transformValue(1))
      case Rotate => AffineTransform.getRotateInstance(transformValue(0), transformValue(1), transformValue(2))
      case Scale => AffineTransform.getScaleInstance(transformValue(0), transformValue(1))
      case Shear => AffineTransform.getShearInstance(transformValue(0), transformValue(1))
    }
  }
}
```

The IDE interface includes a Package Explorer on the left, an Outline on the right, and a Console/Problems area at the bottom. The status bar at the bottom shows "Writable", "Smart Insert", and "16:52".

Regular Expressions

- Triple quote string literals
- The r method on String
- Work as a pattern
- Combine with for loop
- Skip non-matches

```
package scalabook.drawing

import java.awt.Graphics2D
import java.awt.geom.AffineTransform
import swing._
import event._

/**
 * @param parent the DrawTransform that is the parent of this Drawable.
 */
class DrawTransform(parent: DrawTransform) extends Drawable {
  private val subnodes = mutable.Buffer[Drawable]()
  private val transformType = TransformType
  private object TransformType extends Enumeration {
    val Translate, Rotate, Scale, Shear = Value
  }
  import TransformType._
  private val transformValue = Array.fill(3)(0.)

  /**
   * This method applies a transform and draws all the subnodes.
   * @param g the graphics context to draw to.
   */
  def draw(g: Graphics2D) {
    val oldTrans = g.getTransform
    g.transform(buildTransform)
    subnodes.foreach(_.draw(g))
    g.setTransform(oldTrans)
  }

  private def buildTransform: AffineTransform = transformType match {
    case Translate => AffineTransform.getTranslateInstance(transformValue(0), transformValue(1))
    case Rotate => AffineTransform.getRotateInstance(transformValue(0), transformValue(1), transformValue(2))
    case Scale => AffineTransform.getScaleInstance(transformValue(0), transformValue(1))
    case Shear => AffineTransform.getShearInstance(transformValue(0), transformValue(1))
  }
}
```

Package Explorer

Student.scala | DrawingMain.scala | DrawLeaf.scala | DrawTransform.scala | DrawRectangle.scala

Outline

- scalabook.drawing
- import declarations
- DrawTransform
 - parent: scalabook.drawing.DrawTransform
 - propPanel: scala.swing.Component
 - TransformType
 - import declarations
 - transformType: DrawTransform.this.TransformType
 - transformValue: Array[Double]
 - draw(g): java.awt.Graphics2D: Unit
 - buildTransform: java.awt.geom.AffineTransform
 - propertiesPanel(): scala.swing.Component
 - children(): java.util.Enumeration[scalabook.drawing.Drawable]
 - getAllowsChildren(): Boolean
 - getChildAt(childIndex: Int): scalabook.drawing.Drawable
 - getChildCount(): Int
 - getIndex(node: javax.swing.tree.TreeNode): Int
 - getParent(): scalabook.drawing.DrawTransform
 - isLeaf(): Boolean
 - addChild(c: scalabook.drawing.Drawable): Unit
 - removeChild(c: scalabook.drawing.Drawable): Unit
 - toString(): java.lang.String

Problems | Tasks | Console | Error Log

No consoles to display at this time.

Sign in to Google... | Writable | Smart Insert | 16:52

XML

- Literals
- Pattern matching
- Loads DOM
- Xpath style searching
 - \ for immediate contents
 - \\ for deep search
 - Use @ for properties

```
package scalabook.drawing

import java.awt.Graphics2D
import javax.swing.tree.TreeNode
import scala.collection.mutable
import swing._
import event._
import java.awt.geom.AffineTransform

class DrawTransform(parent: DrawTransform) extends Drawable {
  private val subnodes = mutable.Buffer[Drawable]()
  private var propPanel: Component = null

  private object TransformType extends Enumeration {
    val Translate, Rotate, Scale, Shear = Value
  }
  import TransformType._

  private val transformValue = Array.fill(3)(0)

  /**
   * This method applies a transform and draws all the subnodes.
   */
  def draw(g: Graphics2D) {
    val oldTrans = g.getTransform()
    for (subnode <- subnodes)
      subnode.draw(g, transform(oldTrans))
  }

  private def buildTransform: AffineTransform = transformType match {
    case Translate => AffineTransform.getInstance(transformValue(0), transformValue(1))
    case Rotate => AffineTransform.getInstance(transformValue(0), transformValue(1), transformValue(2))
    case Scale => AffineTransform.getScaleInstance(transformValue(0), transformValue(1))
    case Shear => AffineTransform.getShearInstance(transformValue(0), transformValue(1))
  }
}
```

The screenshot shows an IDE with several tabs: Student.scala, DrawingMain.scala, DrawLeaf.scala, DrawTransform.scala, and DrawRectangle.scala. The main editor displays the Scala code for the DrawTransform class. The Outline pane on the right shows the class hierarchy and members. The bottom status bar indicates 'Writable', 'Smart Insert', and the time '16:52'.

Combinatorial Parsers

- CF grammar \rightarrow parser
- $\wedge\wedge$ to specify return
- Quick to parse trees

```
package scalabook.drawing

import java.awt.Graphics2D
import java.awt.geom.AffineTransform
import java.awt.geom.Transform
import java.awt.geom.TransformType
import java.awt.geom.AffineTransform

class DrawTransform(parent: DrawTransform) extends Drawable {
  private val subnodes = mutable.Buffer[Drawable]()

  private object TransformType extends Enumeration {
    val Translate, Rotate, Scale, Shear = Value
  }
  import TransformType._

  private var transformType = Translate
  private val transformValue = Array.fill(3)(0.0)

  /**
   * This method applies a transform and draws all the subnodes.
   * @param g the Graphics2D object to draw to.
   */
  def draw(g: Graphics2D) {
    val oldTrans = g.getTransform
    g.transform(buildTransform)
    subnodes.foreach(_.draw(g))
    g.setTransform(oldTrans)
  }

  private def buildTransform: AffineTransform = transformType match {
    case Translate => AffineTransform.getTranslateInstance(transformValue(0), transformValue(1))
    case Rotate => AffineTransform.getRotateInstance(transformValue(0), transformValue(1), transformValue(2))
    case Scale => AffineTransform.getScaleInstance(transformValue(0), transformValue(1))
    case Shear => AffineTransform.getShearInstance(transformValue(0), transformValue(1))
  }
}
```

Package Explorer: ACMCode, ACSIntern, AdvAlgorithms, AssignmentSubmission, Boogaloo, CSCI1321-511, EffectiveOO, EffectiveOO-F05, EffectiveOO-F05Java, EffectiveOO-F07, HandoutCode, HeaderApplet, icP, MAS, MovieMaker, NavigatorApplet, OctreeRayTracer, PAD2Project, PDAFall03-C++, PDAFall03-Java, PED07, ProfileProject, ProgCompSubmit, RingsVersion3, RingsVersion3_OMP, RoomPlacement, ScalaBook, ScalaFoot, ScalaMAS, ScalaTest2_B, ScalaTreasureHunt, ScalaVis, ScheduleMaker, Servers.

Outline: scalabook.drawing, import declarations, DrawTransform, parent: scalabook.drawing.DrawTransform, propPanel: scala.swing.Component, TransformType, import declarations, transformType: DrawTransform.this.TransformType, transformValue: Array[Double], draw(g): java.awt.Graphics2D: Unit, buildTransform: java.awt.geom.AffineTransform, propertiesPanel(): scala.swing.Component, children(): java.util.Enumeration[scalabook.drawing.Drawable], getAllowsChildren(): Boolean, getChildAt(childIndex: Int): scalabook.drawing.Drawable, getChildCount(): Int, getIndex(node: javax.swing.tree.TreeNode): Int, getParent(): scalabook.drawing.DrawTransform, isLeaf(): Boolean, addChild(c: scalabook.drawing.Drawable): Unit, removeChild(c: scalabook.drawing.Drawable): Unit, toString(): java.lang.String.

Problems, Tasks, Console, Error Log: No consoles to display at this time.

Sign in to Google... Writable Smart Insert 16:52

Parallelism

- Full access to Java libs
 - Functional makes it easier
- Parallel collections
 - Added in 2.9
 - Call par method
 - Fast conversions, $O(1)$
 - Uses work stealing
 - Works with for loops

The screenshot shows an IDE with several Scala files open. The main editor displays the following code:

```
package scalabook.drawing

import java.awt.Graphics2D
import java.awt.geom.AffineTransform
import swing._
import event._

/** @param parent the DrawTransform that is the parent of this Drawable. */
class DrawTransform(parent: DrawTransform) extends Drawable {
  // ...
}

private object TransformType extends Enumeration {
  val Translate, Rotate, Scale, Shear = Value
}

private var transformType = Translate
private val transformValue = Array.fill(3)(0.0)

// ...

private def buildTransform: AffineTransform = transformType match {
  case Translate => AffineTransform.getTranslateInstance(transformValue(0), transformValue(1))
  case Rotate => AffineTransform.getRotateInstance(transformValue(0), transformValue(1), transformValue(2))
  case Scale => AffineTransform.getScaleInstance(transformValue(0), transformValue(1))
  case Shear => AffineTransform.getShearInstance(transformValue(0), transformValue(1))
}

// ...

subnodes.foreach(_ draw(g))
g.setTransform(oldTrans)

// ...

g.transform(buildTransform)
subnodes.foreach(_ draw(g))
g.setTransform(oldTrans)
```

The IDE interface includes a Package Explorer on the left, an Outline on the right, and a console at the bottom.

More Parallel

Actors

- Communicate through messages
- Single threaded in an actor
- Scala Actors
- Akka

The screenshot shows an IDE with several Scala files open. The main editor displays the code for `scalabook.drawing.DrawTransform`. The code defines a class that extends `Drawable` and implements a recursive drawing function. The `draw` method iterates over subnodes, applying a transform and drawing each. The `buildTransform` method uses a match statement to create the appropriate `AffineTransform` based on the `TransformType`.

```
package scalabook.drawing

import java.awt.Graphics2D
import javax.swing.tree.TreeNode
import scala.collection.mutable
import swing._
import event._
import java.awt.geom.AffineTransform

/**
 * This class represents a transformation with children
 */
class DrawTransform(parent: DrawTransform) extends Drawable {
  private val subnodes = mutable.Buffer[Drawable]()
  private object TransformType extends Enumeration {
    val Translate, Rotate, Scale, Shear = Value
  }
  import TransformType._
  private val transformType = Translate
  private val transformValue = Array.fill(3)(0.0)

  /**
   * This method applies a transform and draws all the subnodes.
   *
   * @param g the Graphics2D object to draw to.
   */
  def draw(g: Graphics2D) {
    val oldTrans = g.getTransform
    g.transform(buildTransform)
    subnodes.foreach(_ draw(g))
    g.setTransform(oldTrans)
  }

  private def buildTransform: AffineTransform = transformType match {
    case Translate => AffineTransform.getTranslateInstance(transformValue(0), transformValue(1))
    case Rotate => AffineTransform.getRotateInstance(transformValue(0), transformValue(1), transformValue(2))
    case Scale => AffineTransform.getScaleInstance(transformValue(0), transformValue(1))
    case Shear => AffineTransform.getShearInstance(transformValue(0), transformValue(1))
  }
}
```

The IDE interface includes a Package Explorer on the left, an Outline on the right, and a Problems/Console/Error Log panel at the bottom. The status bar at the bottom shows "Writable", "Smart Insert", and the time "16:52".

DSLs

- Libraries look like language features
- Pass-by-name semantics
- Use implicits for built-in types
- Can use combinatorial parsers

The screenshot shows an IDE with several Scala files open. The main editor displays the following code:

```
package scalabook.drawing

import java.awt.Graphics2D
import java.awt.geom.AffineTransform
import swing._
import event._
import java.awt.geom.AffineTransform

/**
 * @param parent the DrawTransform that is the parent of this Drawable.
 */
class DrawTransform(parent: DrawTransform) extends Drawable {
  private val subnodes = mutable.Buffer[Drawable]()

  private object TransformType extends Enumeration {
    val Translate, Rotate, Scale, Shear = Value
  }
  import TransformType

  private val transformValue = Array.fill(3)(0.0)

  /**
   * This method applies a transform and draws all the subnodes.
   *
   * @param g the Graphics2D object to draw to.
   */
  def draw(g: Graphics2D) {
    val oldTrans = g.getTransform
    g.transform(buildTransform)
    subnodes.foreach(_ draw(g))
    g.setTransform(oldTrans)
  }

  private def buildTransform: AffineTransform = transformType match {
    case Translate => AffineTransform.getTranslateInstance(transformValue(0), transformValue(1))
    case Rotate => AffineTransform.getRotateInstance(transformValue(0), transformValue(1), transformValue(2))
    case Scale => AffineTransform.getScaleInstance(transformValue(0), transformValue(1))
    case Shear => AffineTransform.getShearInstance(transformValue(0), transformValue(1))
  }
}
```

The IDE interface includes a Package Explorer on the left, an Outline on the right, and a Problems/Console/Error Log panel at the bottom. The status bar at the bottom shows "Writable", "Smart Insert", and "16:52".

Web Frameworks

- Lift

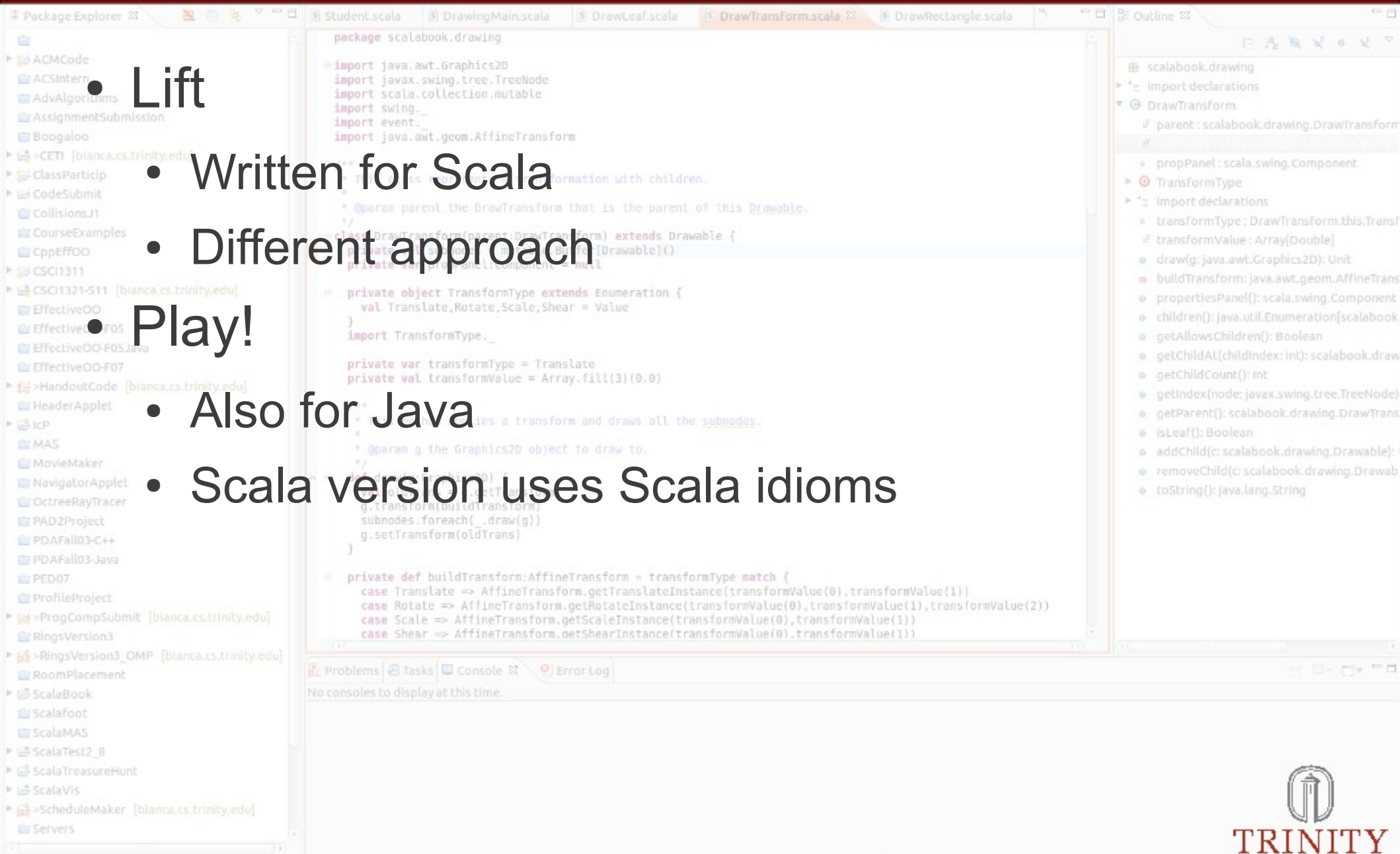
- Written for Scala

- Different approach

- Play!

- Also for Java

- Scala version uses Scala idioms



Conclusions

• Get most of the best of all worlds

- Less boiler plate
- Static type safety and speed
- High expressivity

• Keep current JVM functionality

• Easier parallel

• High level libraries

• Look like language features

• Easy to use

• DSLs

