

# First Programming Language in CS Education – The Arguments for Scala

WORLD COMP 2011

By

Dr. Mark C. Lewis  
Trinity University



TRINITY  
UNIVERSITY

```
/* 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: List[Drawable] = List()
  private var transformType = Translate
  private val transformValue = Array.fill(3)(0.0)

  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)
  }

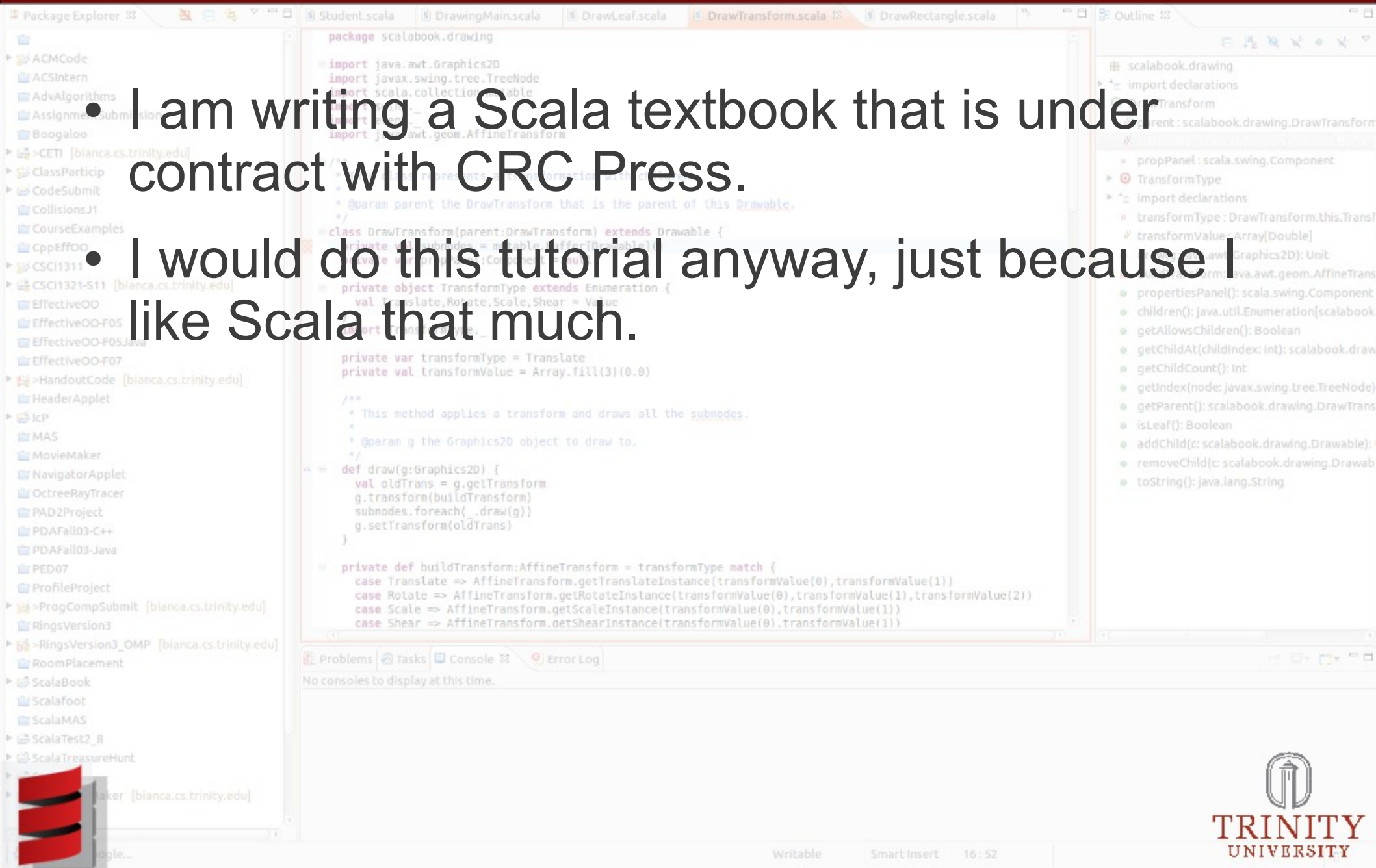
  private def buildTransform(): AffineTransform {
    case Translate => AffineTransform.getTranslateInstance(transformValue(0), transformValue(1))
    case Rotate => AffineTransform.getRotateInstance(transformValue(0), transformValue(1), transformValue(2))
    case Scale => AffineTransform.getScaleInstance(transformValue(0),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

- ClassParticip
- CodeSubmit
- CollisionsJ1
- CourseExamples
- CppEffOO
- CSCI1311
- CSCI1321-511 [bianca.cs.trinity.edu]
- EffectiveOO
- EffectiveOO-F05
- EffectiveOO-F05Java
- EffectiveOO-F07
- >HandoutCode [bianca.cs.trinity.edu]
- HeaderApplet
- icP
- MAS
- MovieMaker
- NavigatorApplet
- OctreeRayTracer
- PAD2Project
- PDAFall03-C++
- PDAFall03-Java
- PED07
- ProfileProject
- >ProgCompSubmit [bianca.cs.trinity.edu]
- RingsVersion3
- >RingsVersion3\_OMP [bianca.cs.trinity.edu]
- ScalaVis
- >ScheduleMaker [bianca.cs.trinity.edu]
- Servers

- 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

# Disclaimer

- I am writing a Scala textbook that is under contract with CRC Press.
- I would do this tutorial anyway, just because I like Scala that much.



The screenshot shows an IDE window with several tabs: 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 javax.swing.tree.TreeNode
import scala.collection.mutable
import java.awt.geom.AffineTransform

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

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

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 and methods for 'scalabook.drawing.DrawTransform'.

# 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 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 the following
 * properties:
 * - transformType: DrawTransform.this.TransformType
 * - transformValue: Array[Double]
 */
class DrawTransform(parent: DrawTransform) extends Drawable {
  // ... (code omitted) ...

  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)

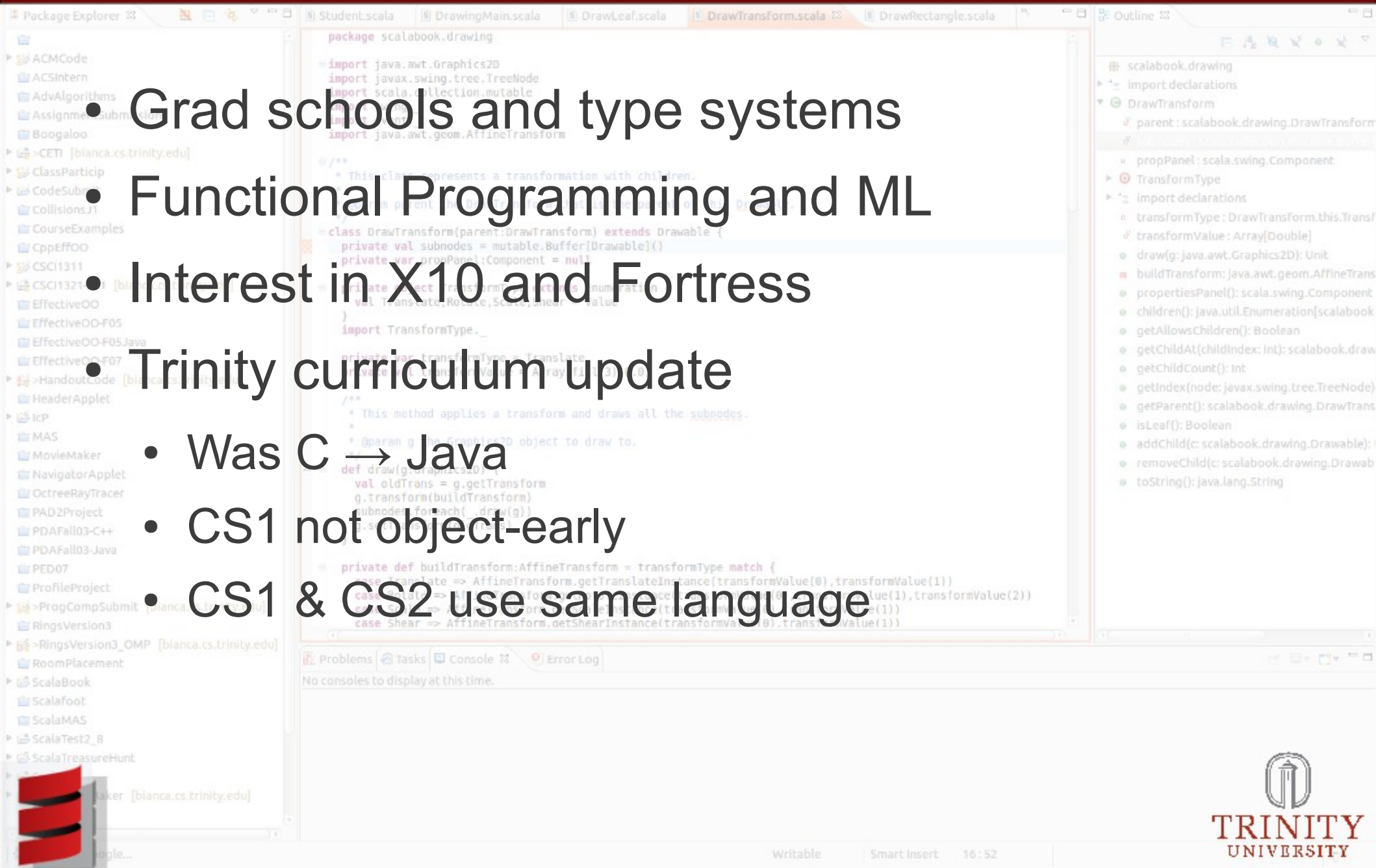
  /**
   * This method applies a transform and draws all the subnodes.
   *
   * @param g the Graphics2D object to draw to.
   */
  def draw(g: Graphics2D) {
    val oldTrans = g.getTransform
    g.transform(buildTransform)
    subnodes.foreach(_.draw(g))
    g.setTransform(oldTrans)
  }

  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 Problems/Console/Error Log panel at the bottom. The status bar at the bottom indicates 'Writable', 'Smart Insert', and the time '16:52'.

# Scala and Me

- Grad schools and type systems
- Functional Programming and ML
- Interest in X10 and Fortress
- Trinity curriculum update
  - Was C  $\rightarrow$  Java
  - CS1 not object-early
  - CS1 & CS2 use same language





# Basics of Scala

## • “Scalable Language”

## • Multi-Paradigm

- Productivity of scripting languages

- Expressivity of functional languages

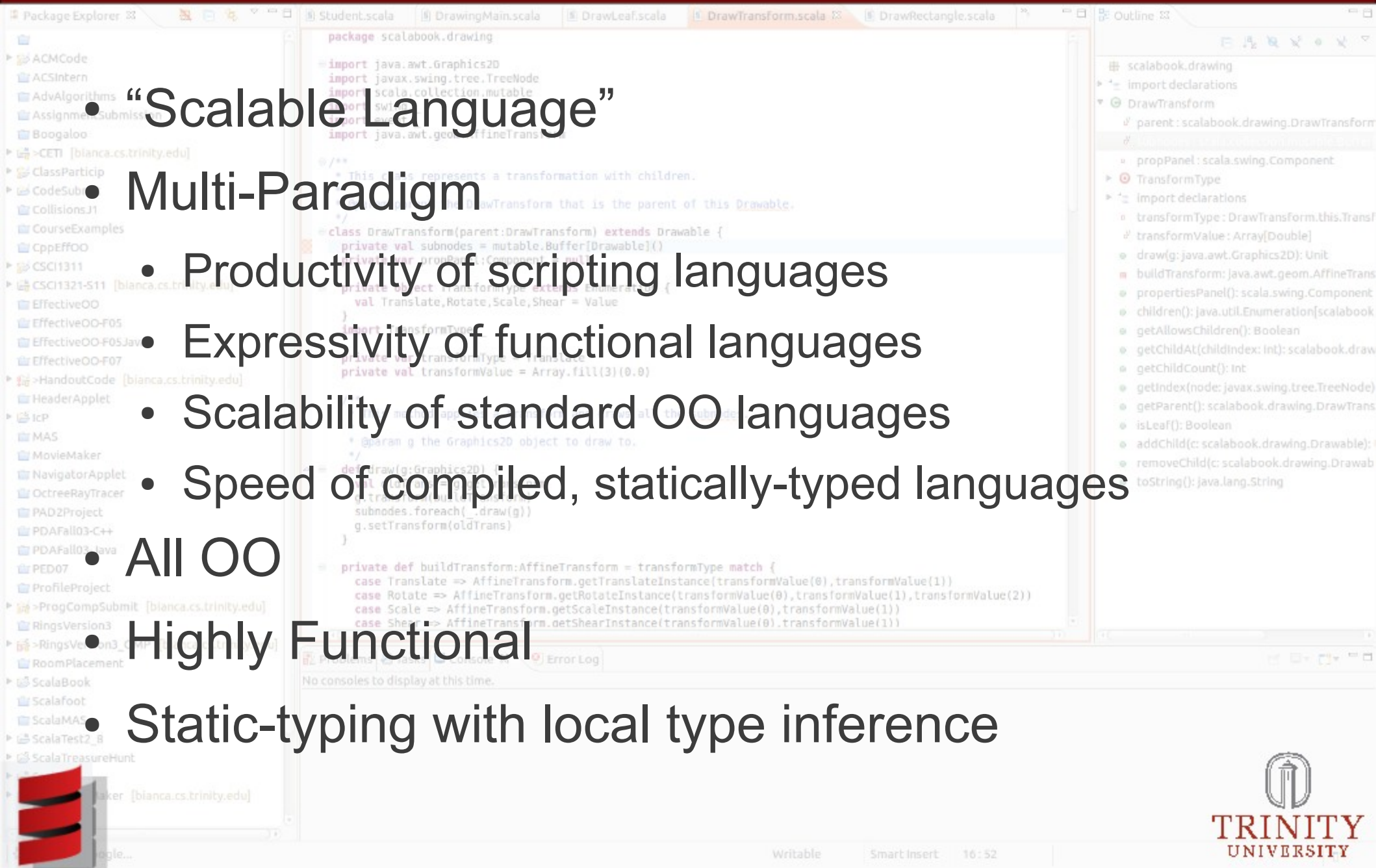
- Scalability of standard OO languages

- Speed of compiled, statically-typed languages

## • All OO

## • Highly Functional

- Static-typing with local type inference



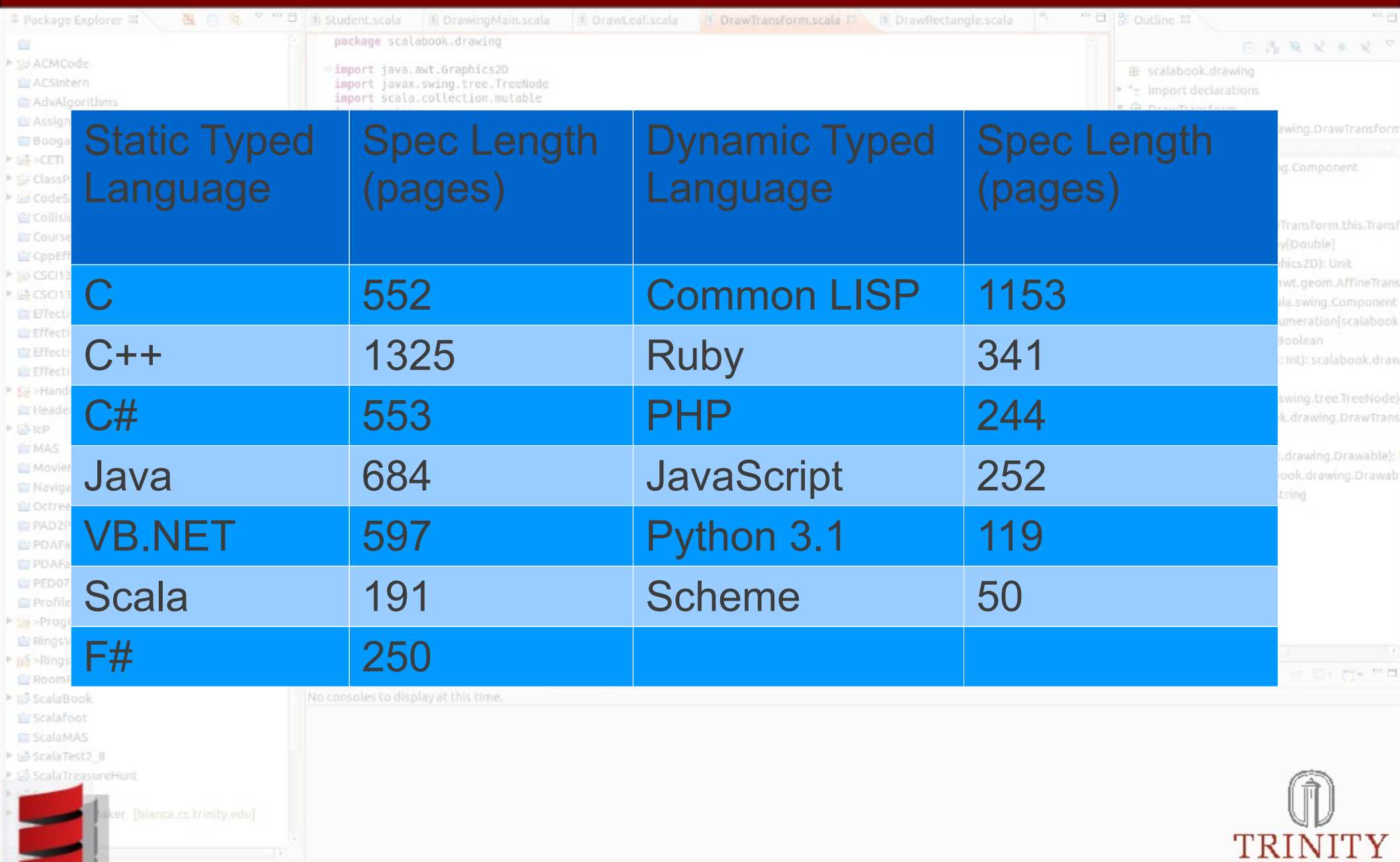
# Too Complex?

## Reasons for perception

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

The screenshot shows an IDE window with several Scala files open. The main editor displays the code for `scalabook.drawing.DrawTransform`. The code includes imports for `java.awt.Graphics2D`, `javax.swing.tree.TreeNode`, `scala.collection.mutable`, and `java.awt.geom.AffineTransform`. It defines a `DrawTransform` class that extends `Drawable` and implements a `draw(g)` method. The `draw` method uses a `match` expression to handle different `TransformType` values: `Translate`, `Rotate`, `Scale`, and `Shear`. The IDE also shows a Package Explorer on the left with a tree view of the project structure, and an Outline on the right showing the class hierarchy and members of `DrawTransform`. The status bar at the bottom indicates 'Writable', 'Smart Insert', and the time '16:52'.

# Shorter Language Specification

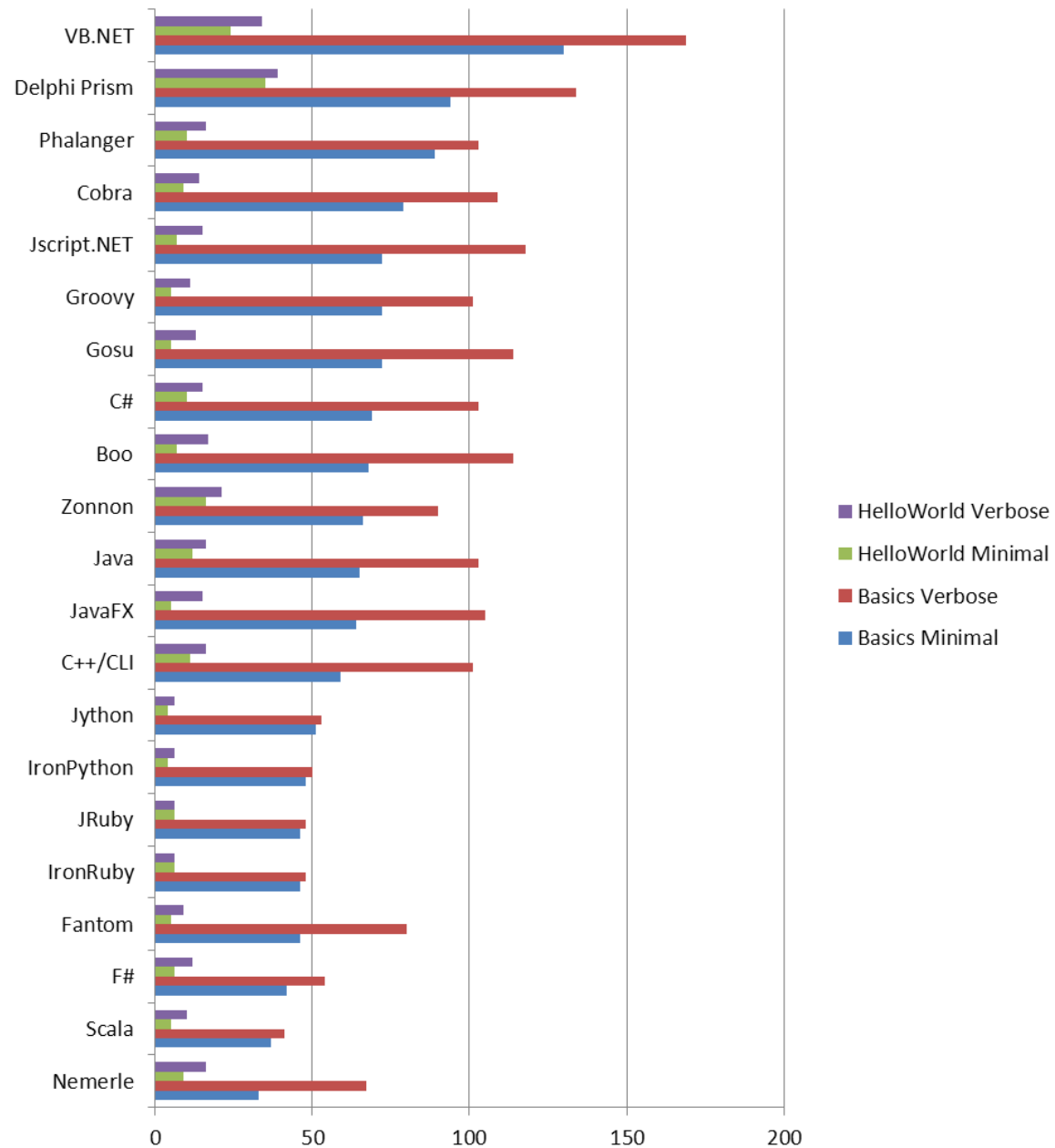
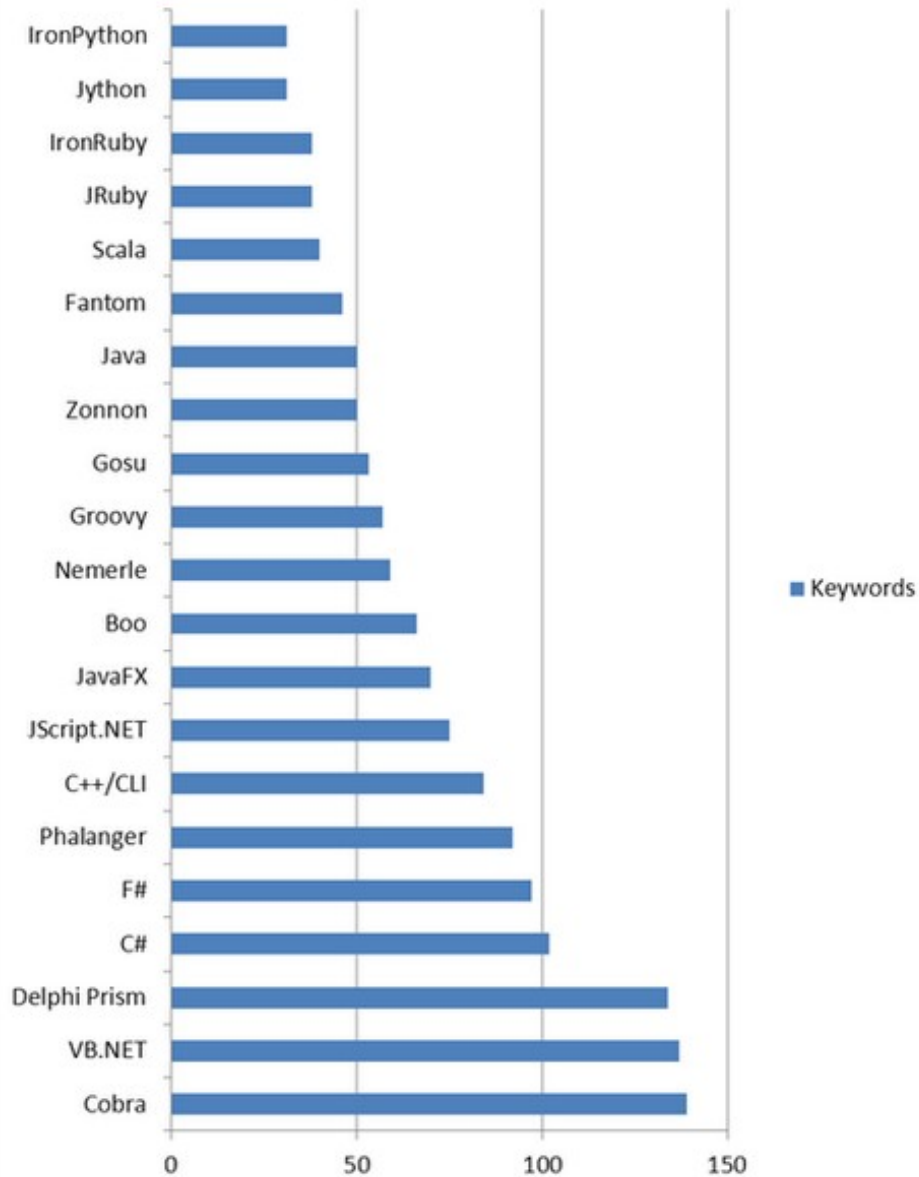


The background image shows an IDE window with several Scala files open: Student.scala, DrawingMain.scala, DrawLeaf.scala, DrawTransform.scala, and DrawRectangle.scala. The code in the active window shows imports for Graphics2D, TreeNode, and mutable. A table is overlaid on the IDE, comparing the specification 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 and light blue cells.

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



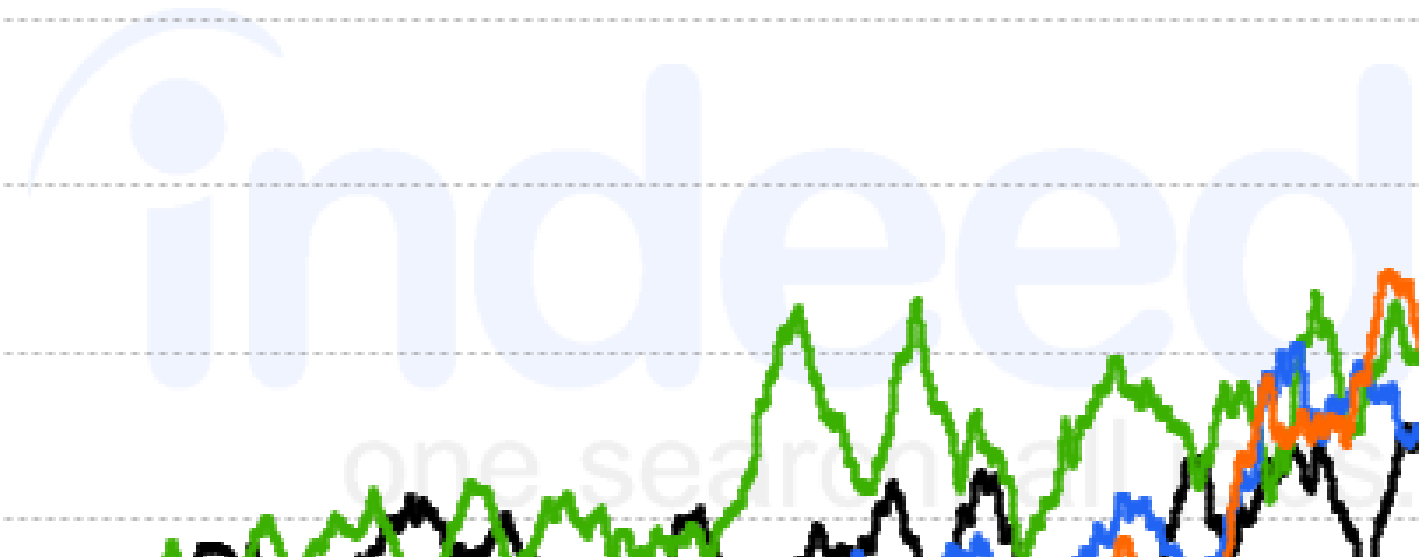
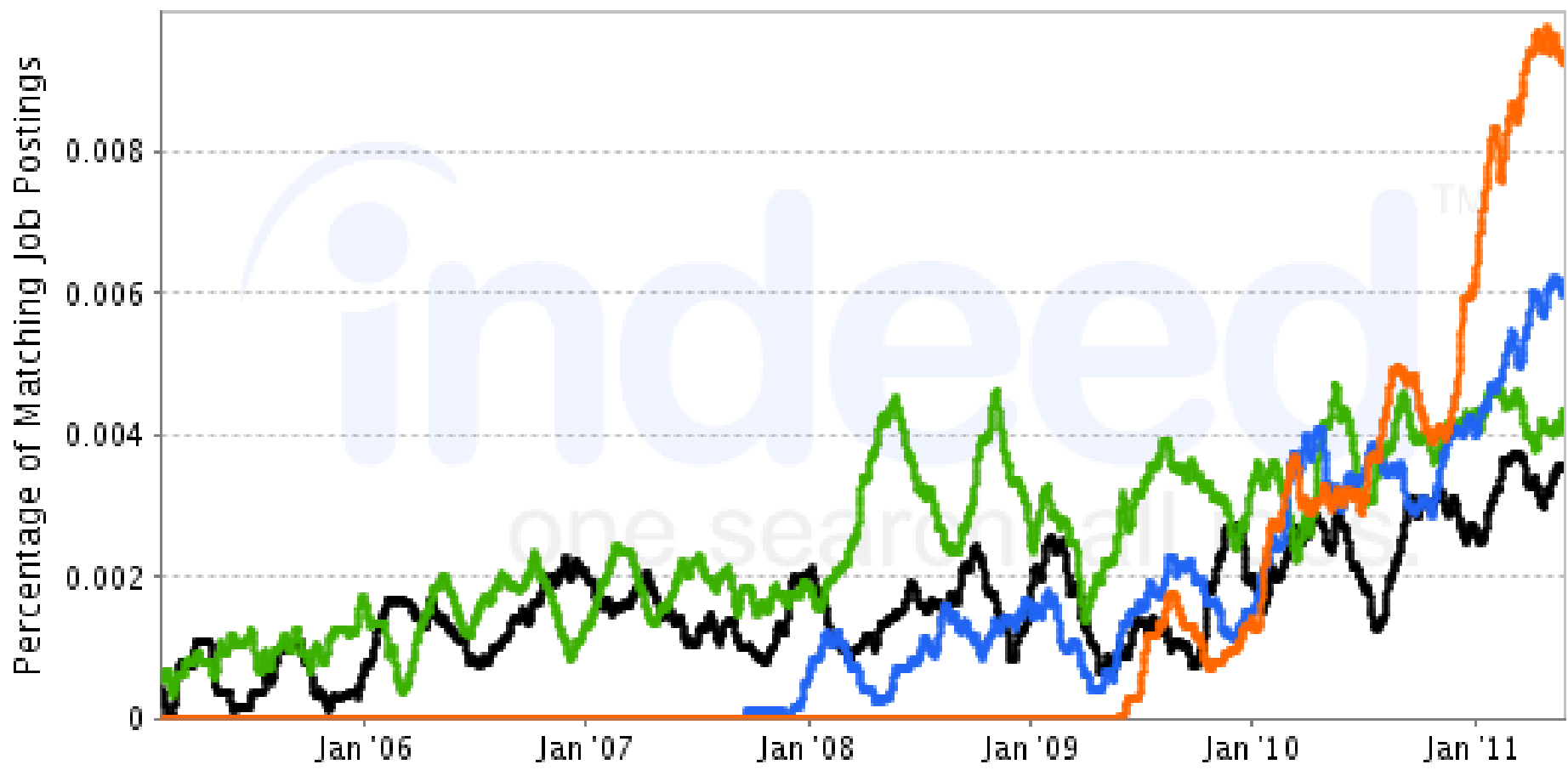


# Jobs



## Job Trends from Indeed.com

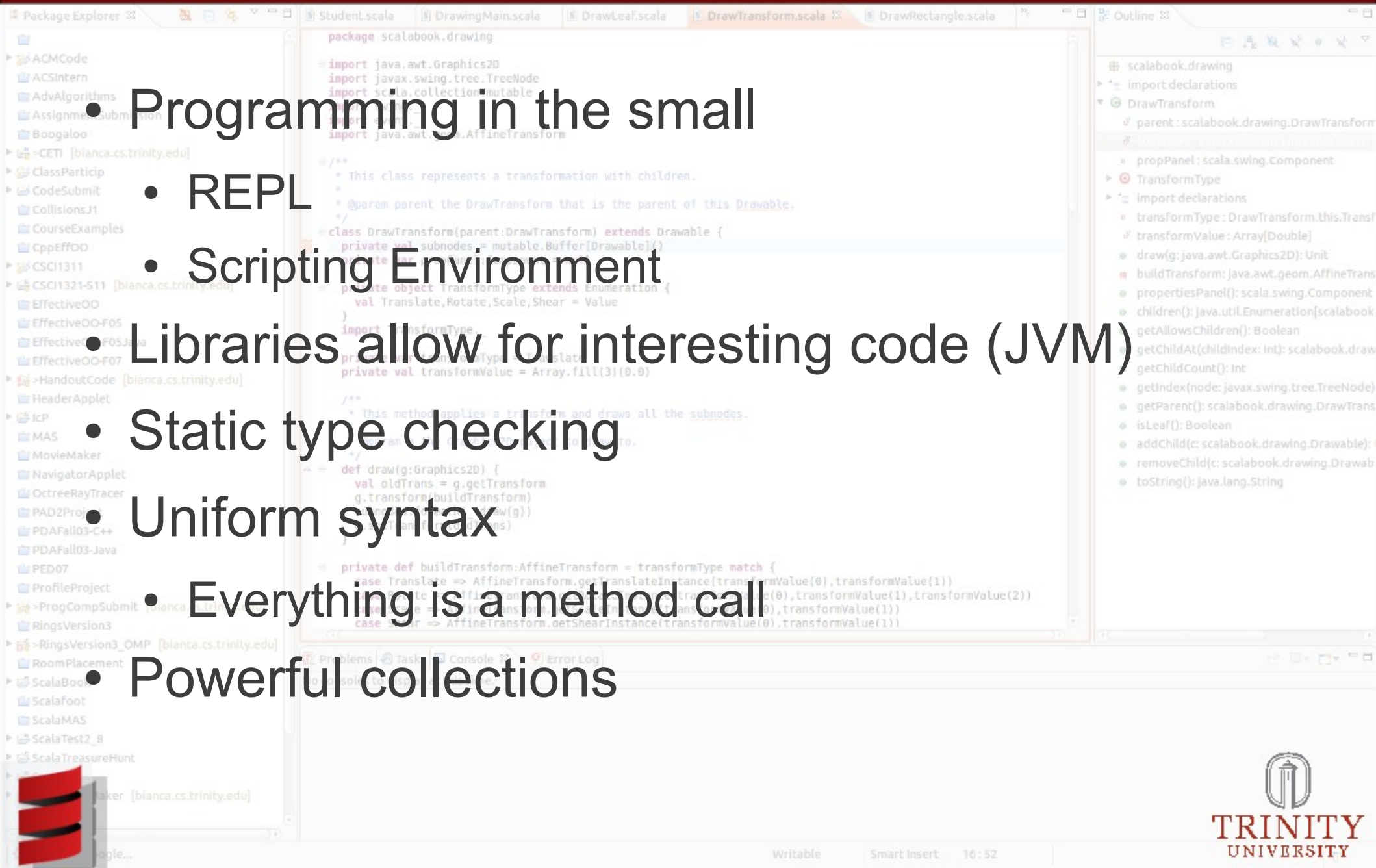
— scala developer — erlang developer — scheme developer — lisp developer



# Benefits in CS 1

## • Programming in the small

- REPL
- Scripting Environment
- Libraries allow for interesting code (JVM)
- Static type checking
- Uniform syntax
- Everything is a method call
- Powerful collections



# Starting Off

• “Hello, World” and such

• REPL

• Types - “lack” of primitives  
- tuples

• Statements/expressions

• Semicolon inference

• Simple input

• Variables

• val

• var

• Scripts

```
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.
 */
abstract class Drawable {
  private val parentNode: TreeNode
  private var propPanel: Component = null

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

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

  /**
   * Draw this Drawable 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)
  }

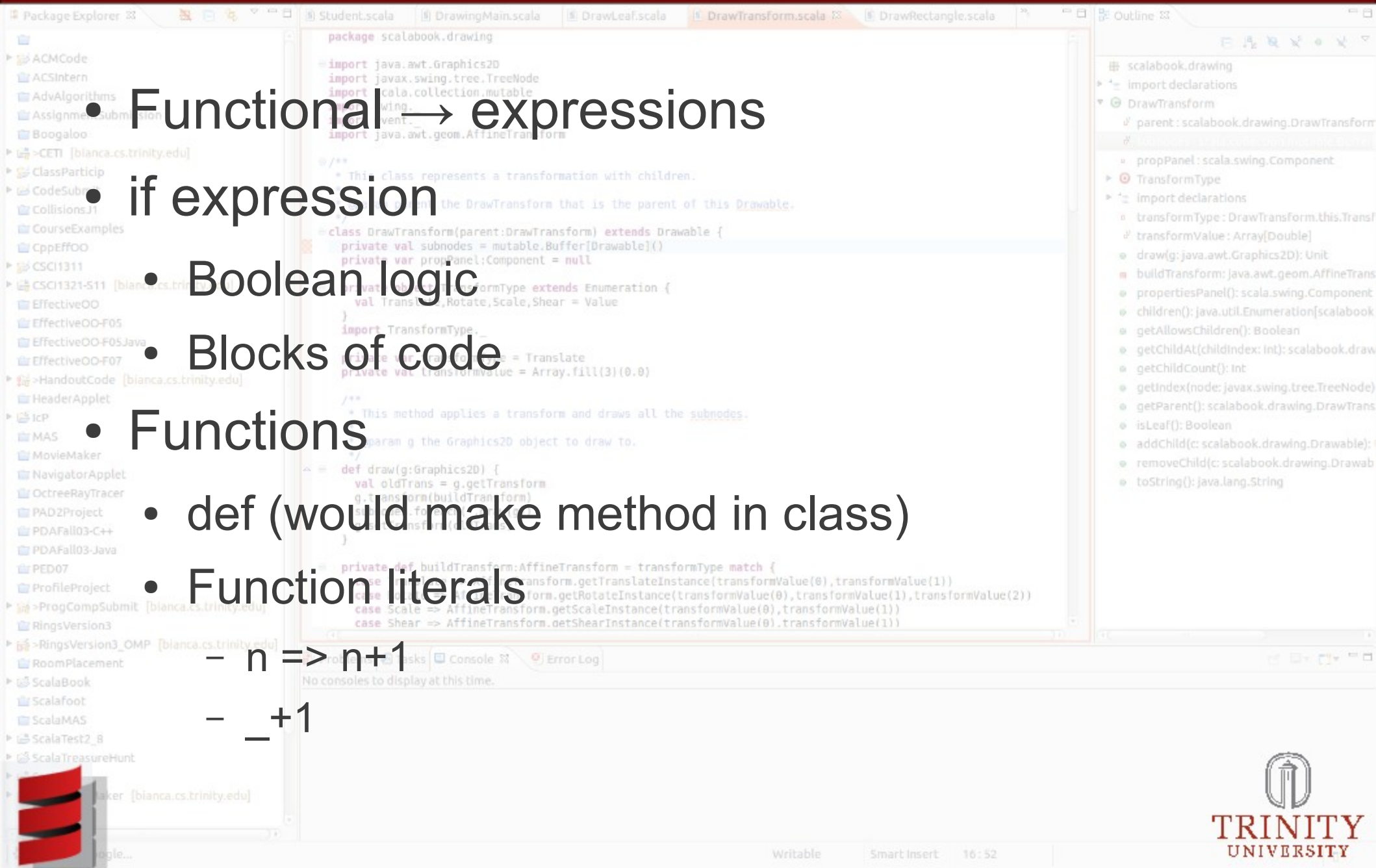
  private def buildTransform: AffineTransform = transformType match {
    case Translate => AffineTransform.getTranslateInstance(transformValue(0), transformValue(1))
    case Rotate => AffineTransform.getRotateInstance(transformValue(0), transformValue(1), transformValue(2))
    case Scale => AffineTransform.getScaleInstance(transformValue(0),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

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

# Conditions and Functions

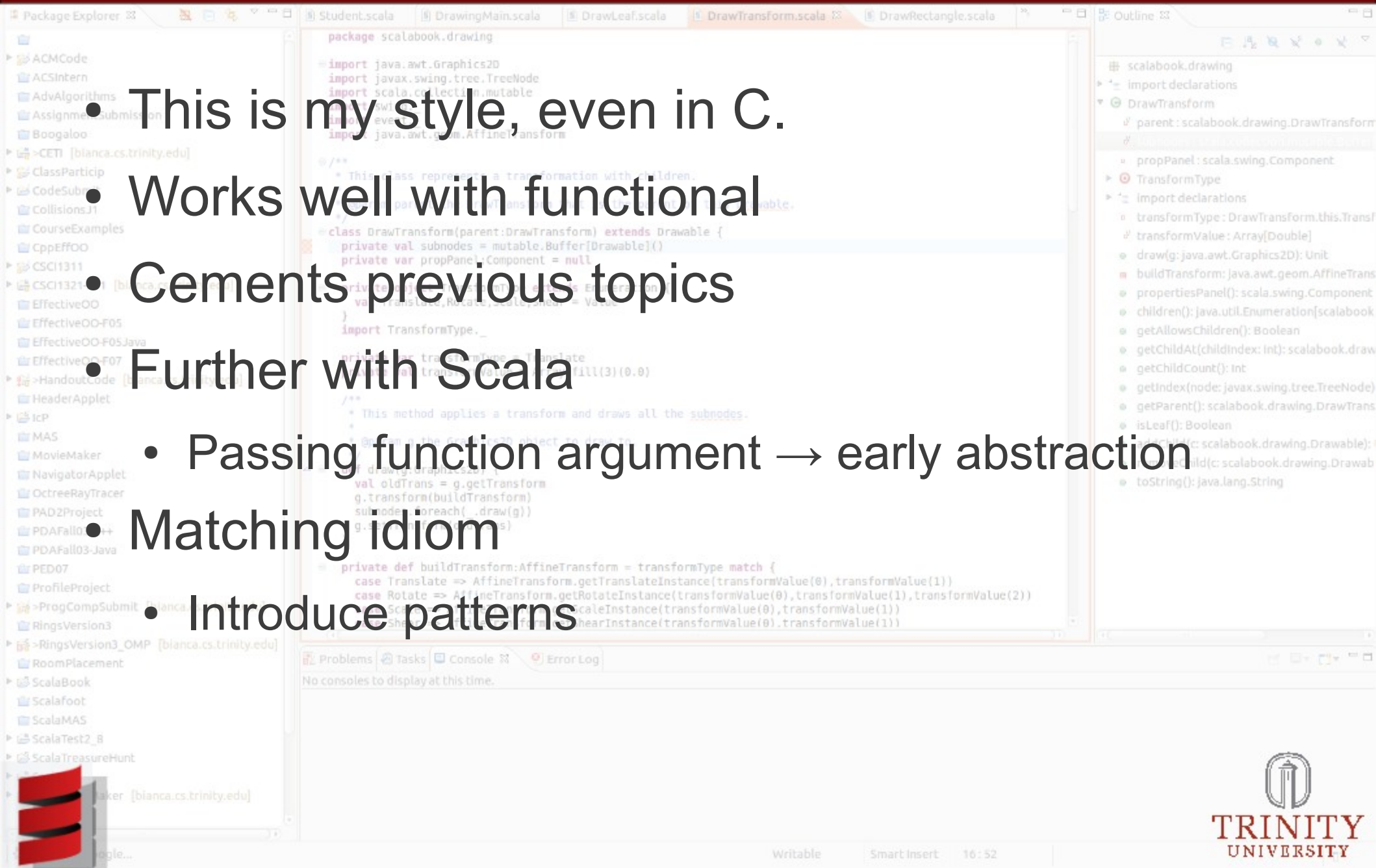
- Functional  $\rightarrow$  expressions
- if expression
- Boolean logic
- Blocks of code
- Functions
  - def (would make method in class)
  - Function literals
    - $n \Rightarrow n+1$
    - $\_ + 1$





# Recursion for Iteration

- This is my style, even in C.
- Works well with functional
- Cements previous topics
- Further with Scala
  - Passing function argument → early abstraction
- Matching idiom
- Introduce patterns



# Collections (Array/List)

## • Just for Scala

- Doesn't make sense before loops in most languages.

## • One mutable, one immutable

## • Many standard methods

## • Many higher-order methods

## • Syntax

- Use () for indexing

- List also have ML style operations

## • Creation, pass-by-name

```
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 this is a child of
 *  @param transformType the type of transformation
 */
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)(0.0)

  /** This method applies a transform and draws all the subnodes
   *  @param g the Graphics2D object to draw on
   *  @param g the Graphics2D object to draw on
   */
  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))
  }
}
```

# Loops

- While loop
  - Not an expression
- For loop
  - Really for-each
  - yield
  - Ranges
  - Many options
    - Multiple generators
    - If guards
    - Variables
    - Patterns

```
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

/**
 * 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
  }

  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))
  }
}
```

# Files

- Can use Scanner
- scala.io.Source
  - Scala Iterator[Char]
  - getLines : Iterator[String]
  - Use with higher-order methods
- Write with PrintWriter
- Introduce APIs?

The screenshot shows an IDE with several Scala files open. The main editor displays the `DrawTransform.scala` file, which defines a `DrawTransform` class extending `Drawable`. The code includes imports for `java.awt.Graphics2D`, `javax.swing.tree.TreeNode`, `scala.collection.mutable`, and `java.awt.geom.AffineTransform`. The class has a `parent` of type `DrawTransform` and a `subnodes` of type `mutable.Buffer[Drawable]`. It implements the `draw` method and has a `buildTransform` method that matches on `TransformType` to create `AffineTransform` instances for translation, rotation, scaling, and shearing.

The Package Explorer on the left shows a project structure with folders like `ACMCode`, `ACSIntern`, and `ScalaBook`. The Outline on the right shows the class hierarchy for `scalabook.drawing`, including `DrawTransform` and its `propPanel` and `TransformType` attributes.



# Case Classes

- Immutable struct in simplest usage
- Simple syntax for grouping data
- Works as a pattern
- Copy method

The screenshot shows an IDE with the following components:

- Package Explorer:** Lists various project folders like ACMCode, ACSIntern, AdvAlgorithms, etc.
- Code Editor:** Displays the source code for `scalabook.drawing.DrawTransform`. The code includes imports for `java.awt.Graphics2D`, `javax.swing.tree.TreeNode`, `scala.collection.mutable`, and `java.awt.geom.AffineTransform`. It defines a case class `DrawTransform` with a `parent` of type `DrawTransform` and a `transformType` of type `TransformType`. The class has a `draw` method that applies the transform and draws subnodes, and a `buildTransform` method that matches the `transformType` to create an `AffineTransform`.
- Outline:** Shows the structure of the `scalabook.drawing` package, including `import declarations`, `DrawTransform`, `propPanel`, `TransformType`, and `import declarations` for `DrawTransform`.
- Problems/Console/Errors:** Shows "No consoles to display at this time."

# GUIs

- scala.swing wraps javax.swing
- Cleaner beginner syntax
  - No explicit inheritance
  - Reactions use partial functions
- Drawbacks
  - Currently no JTree
  - Tables complex
  - Button syntax uses companion object

The screenshot shows an IDE with several Scala files open. The main file is `DrawTransform.scala` in the `scalabook.drawing` package. The code defines a `DrawTransform` class that extends `Drawable` and implements a `draw` method. It uses `TransformType` to handle different transformations like Translate, Rotate, Scale, and Shear. The code also shows a `buildTransform` method that uses a `match` expression to create the appropriate `AffineTransform` instance based on the `transformType`.

```
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 var propPanel: Component = null

  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))
  }
}
```

# Graphics

- Full Java2D

- Really using Java

- Override paint method

- Use BufferedImage

- Events for animations

- Keyboard

- Mouse

- Timer

```
package scalabook.drawing

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

/**
 * This class represent a transformation with children.
 * The parent of this Drawable.
 */
class DrawTransform(parent: DrawTransform) extends Drawable {
  private val subnodes = mutable.Buffer[Drawable]()
  private var transformType: TransformType = Value
  private val Translate, Rotate, Scale, Shear = Value

  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.transform(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))
  }
}
```

# Sorting & Searching

- Monomorphic at this point
- Can write your own visualization
- There are methods in collections

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 children.
 */
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)

  /**
   * 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 Outline view on the right shows the class 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



# Other Stuff

## • XML

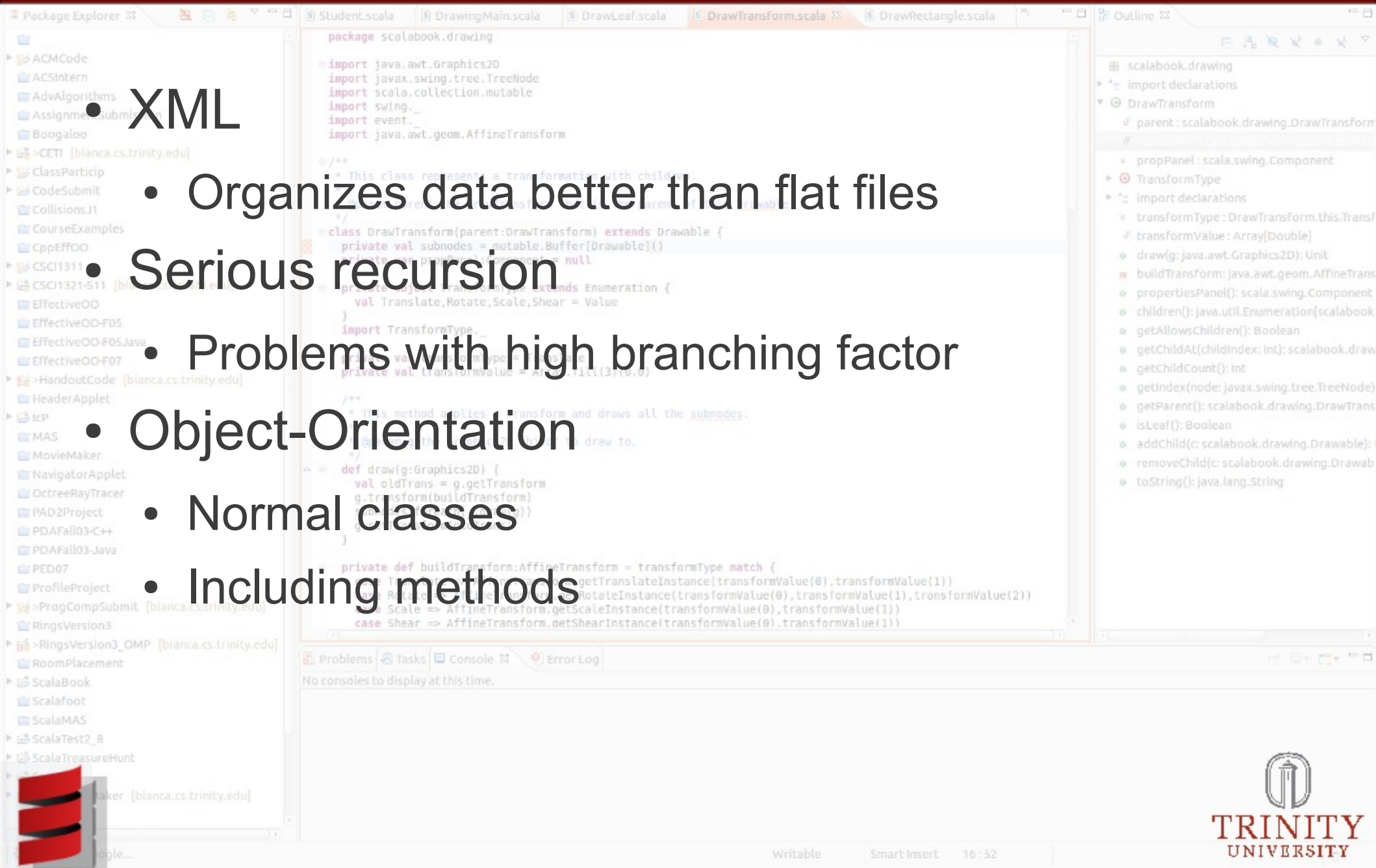
- Organizes data better than flat files

## • Serious recursion

- Problems with high branching factor

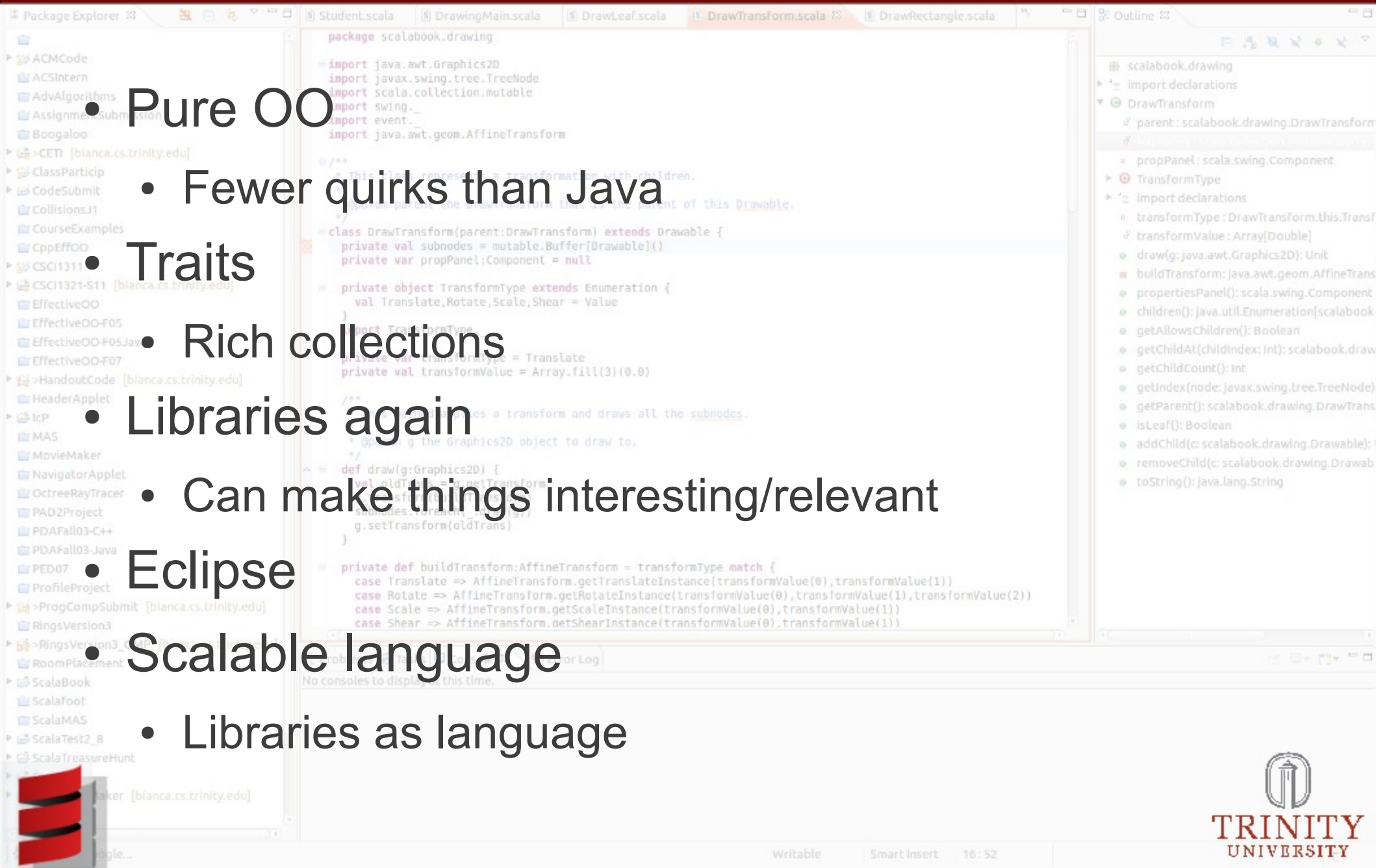
## • Object-Orientation

- Normal classes
- Including methods



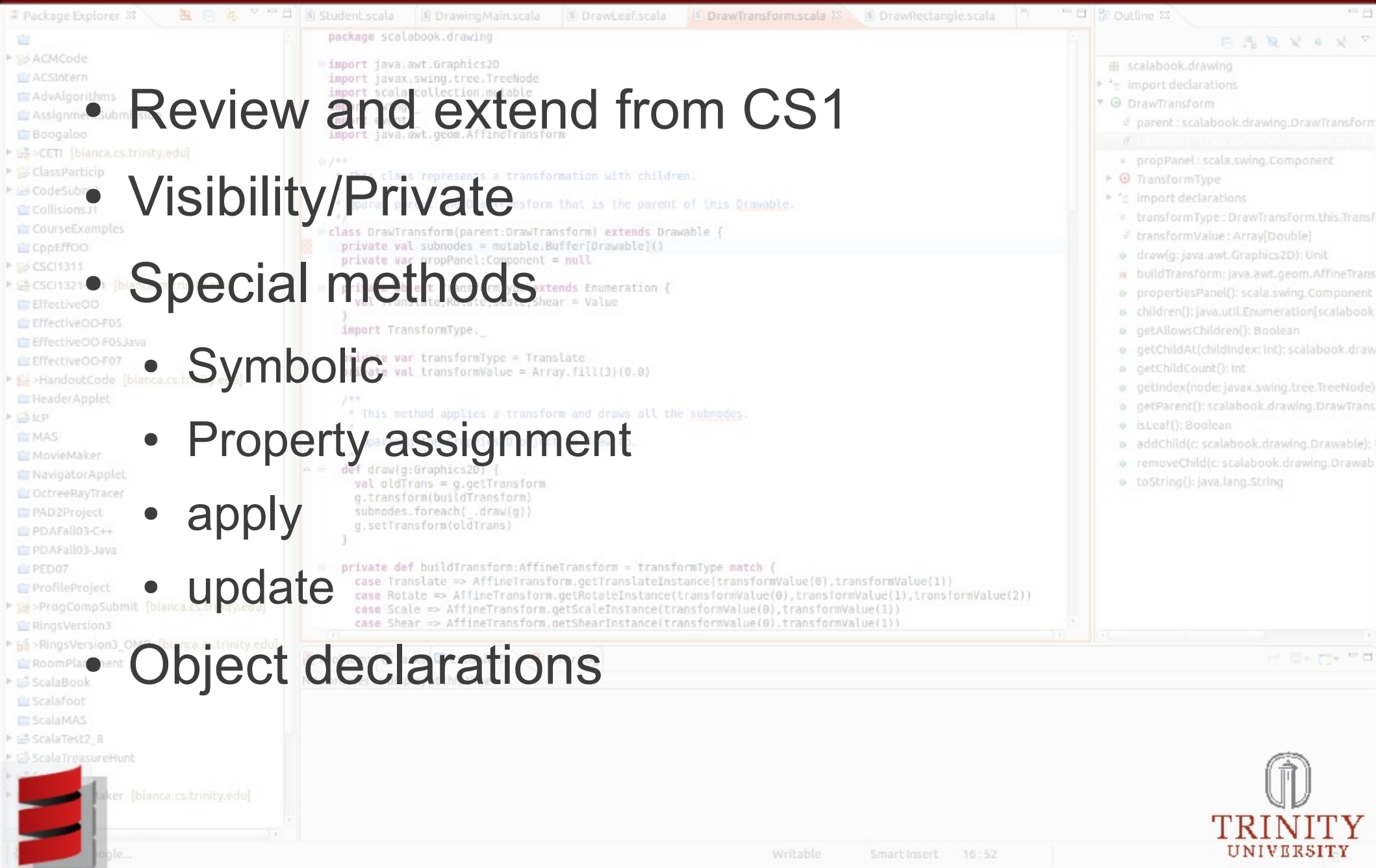
# Benefits in CS 2

- Pure OO
- Fewer quirks than Java
- Traits
- Rich collections
- Libraries again
- Can make things interesting/relevant
- Eclipse
- Scalable language
- Libraries as language



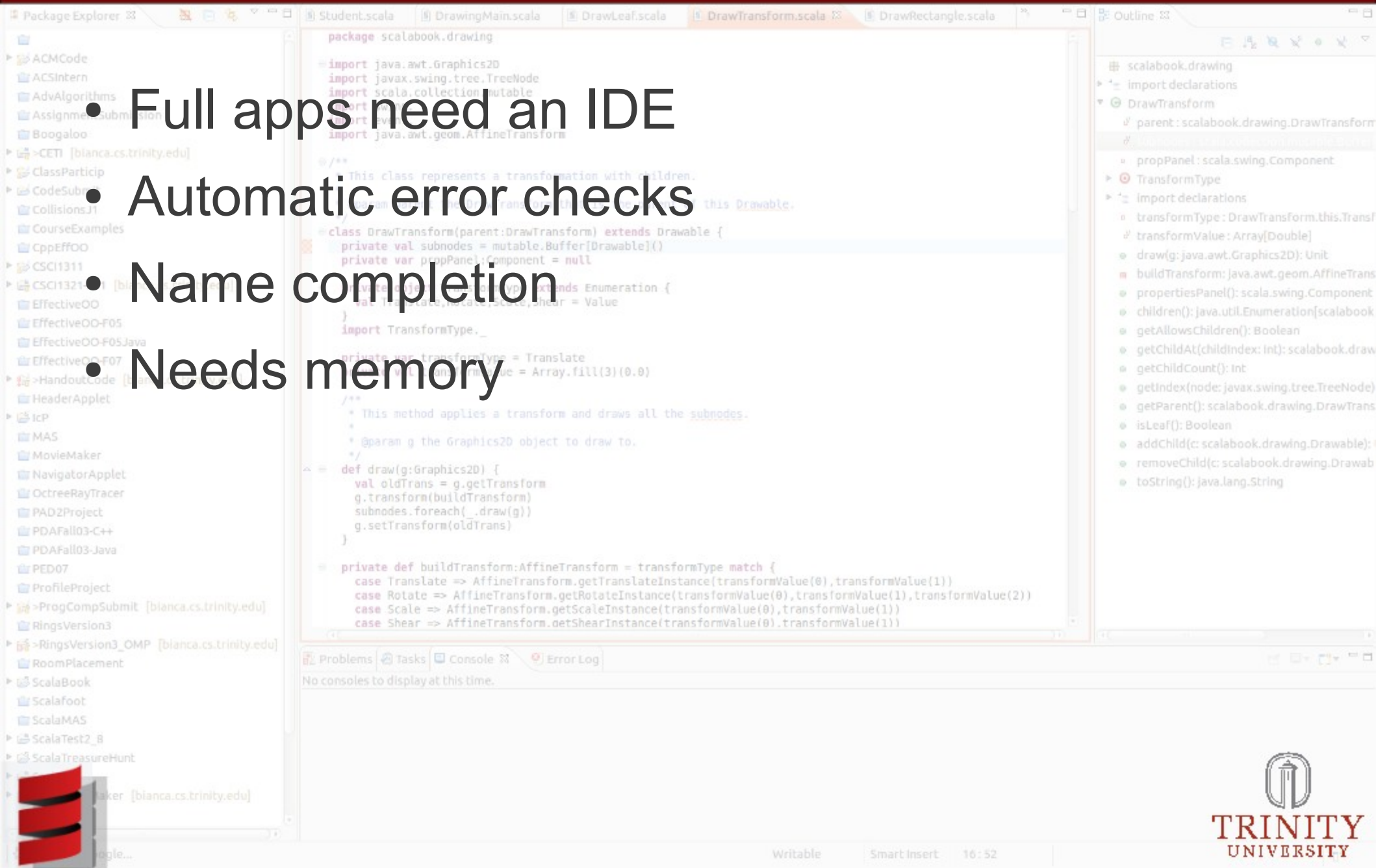
# OO for Larger Programs

- Review and extend from CS1
- Visibility/Private
- Special methods
  - Symbolic
  - Property assignment
  - apply
  - update
- Object declarations



# Eclipse

- Full apps need an IDE
- Automatic error checks
- Name completion
- Needs memory





# Polymorphism

## • Inheritance/Subtyping

- Traits

- Protected visibility

## • Parametric Polymorphism

- Type bounds

- Parametric methods → polymorphic sorts

```
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 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.
   */
  def draw(g: Graphics2D) {
    val oldTrans = g.getTransform
    subnodes.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))
  }
}
```

# Other Collections

## • Sets

- Include expected methods

## • Maps

- (key,value)
- key  $\rightarrow$  value

## • Buffers

- Mutable
- Variable sized

- Many options mutable and 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 transformation with children
 */
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 transformType = Translate
  private val transformValue = Array.fill(3)(0.0)

  /**
   * 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))
  }
}
```

# Multithreading

- Many approaches
  - `java.lang.Thread`
  - `java.util.concurrent`
  - Parallel collections
    - Added in 2.9
  - Actors
- Simplified by functional

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

/**
 * This class represents a transformation with children.
 * The parent of this class is DrawTransform, which 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 represents 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.getInstance(transformValue(0), transformValue(1))
    case Rotate => AffineTransform.getInstance(transformValue(0), transformValue(1), transformValue(2))
    case Scale => AffineTransform.getInstance(transformValue(0), transformValue(1))
    case Shear => AffineTransform.getInstance(transformValue(0), transformValue(1))
  }
}
```

The Outline view on the right shows the class hierarchy and members for `scalabook.drawing.DrawTransform`:

- 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

# Networking

- Use java.net and java.io
- Streams
- Sockets
- Serialization

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 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 var 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))
  }
}
```

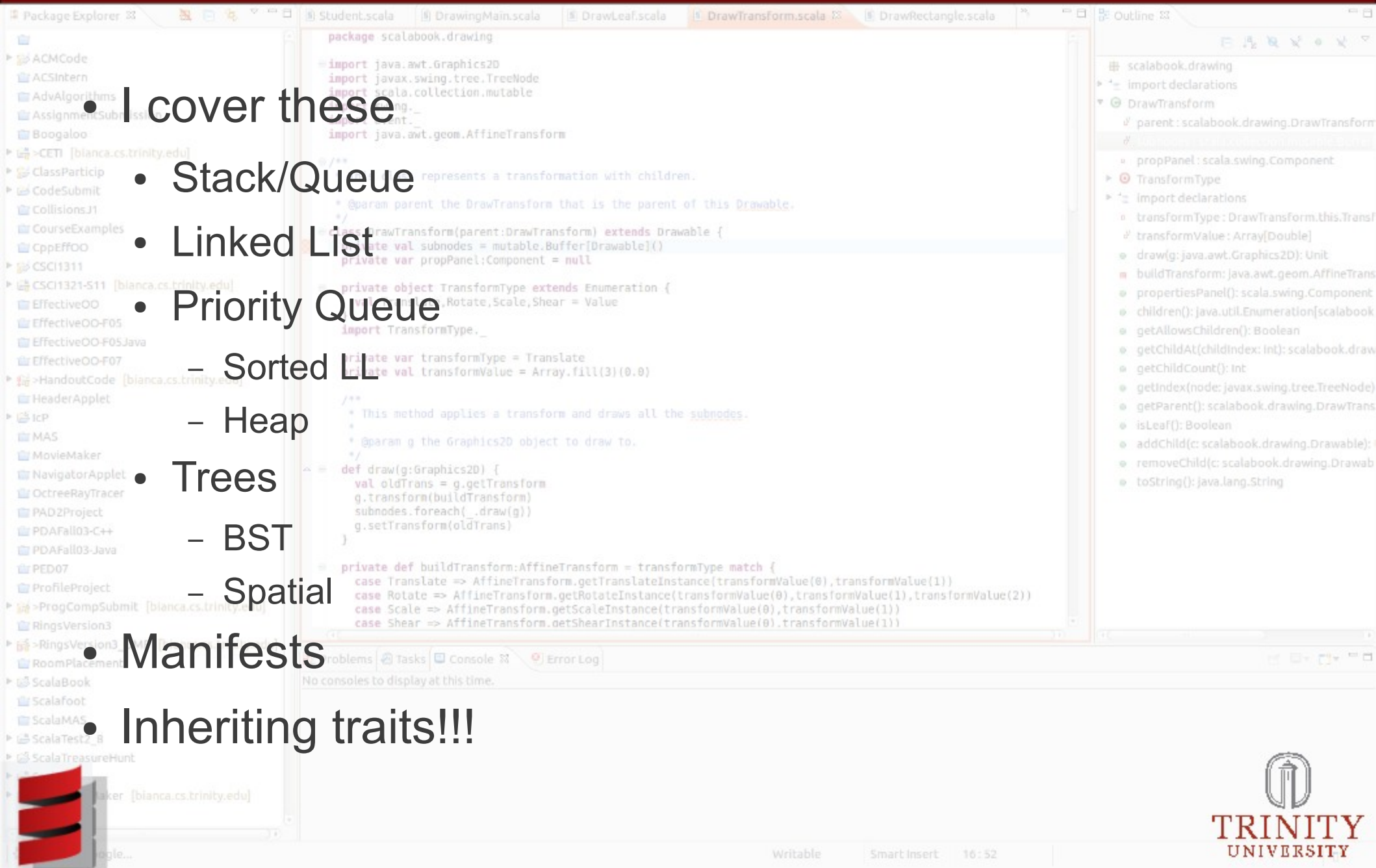
The Outline view on the right shows the class structure:

- 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



# ADTs

- I cover these
  - Stack/Queue
  - Linked List
  - Priority Queue
    - Sorted LL
    - Heap
  - Trees
    - BST
    - Spatial
  - Manifests
  - Inheriting traits!!!



# Grammars

## • RegEx

- .r method on String

- Triple quote strings

- Patterns

## • Combinatorial Parsers

- CF grammar → parser

- Arithmetic example

```
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.
 * The parent of this Drawable.
 */
class DrawTransform(parent: DrawTransform) extends Drawable {
  private val subnodes = mutable.Buffer[Drawable]()
  private val transformType = TransformType
  private var transformValue = Array.fill(3)(0.0)

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



# Beyond CS 1&2

- Akka
- Non-JVM implementations
- GPGPU libraries

The screenshot shows an IDE window with several tabs: 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 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 var propPanel: Component = null

  private val 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 tree of project files and folders, and an Outline on the right showing the class hierarchy for DrawTransform. The status bar at the bottom indicates 'Writable', 'Smart Insert', and the time '16:52'.

# Conclusions

- Good early on
  - REPL
  - Scripting
- Grows well
  - OO & static types
  - IDE support
  - Complete libraries
- Complexity not required

