The Scala Programming Language

Troy Hut and Benjamin Killeen

Introduction

Examples Examples

Natural Numbers

The Scala Programming Language

Troy Hut and Benjamin Killeen

The Scala Programming Language

Troy Hut and Benjamin Killeen

Introduction

Illustrative Examples

Natural Numbers

Scala is:

The Scala Programming Language

Troy Hut and Benjamin Killeen

Introduction

Illustrative Examples

Natural Numbers

Scala is:

■ Object oriented

The Scala Programming Language

Troy Hut and Benjamin Killeen

Introduction

Illustrative Examples

Natural Numbers

Scala is:

- Object oriented
- Functional

The Scala Programming Language

Troy Hut and Benjamin Killeen

Introduction

Illustrative Examples

Natural Numbers

Scala is:

- Object oriented
- Functional
- Statically typed

Scala is Scalable

The Scala Programming Language Troy Hut and

Benjamin Killeen

Introduction

Illustrative

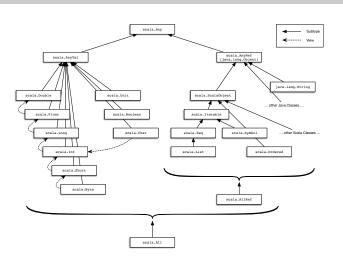


Figure: The Scala class hierarchy.

Traits, Objects, and Classes

```
The Scala
Programming
Language
Troy Hut and
```

Benjamin Killeen

Introductio

Illustrative Examples

```
trait Nat {
  def isZero (): Boolean
  def pred (): Nat
  def succ (): Nat
}
```

Traits, Objects, and Classes

```
The Scala
Programming
Language
```

Troy Hut and Benjamin Killeen

Introduction

Illustrative

```
trait Nat {
  def isZero (): Boolean
  def pred (): Nat
  def succ (): Nat
}

object Z extends Nat {
  def isZero () = true
  def succ () = S(this)
  def pred () = this
}
```

Traits, Objects, and Classes

The Scala Programming Language

Troy Hut and Benjamin Killeen

Introduction

Illustrative Examples

```
trait Nat {
  def isZero (): Boolean
  def pred (): Nat
 def succ (): Nat
object Z extends Nat {
  def isZero () = true
  def succ () = S(this)
  def pred () = this
}
case class S(v: Nat) extends Nat {
  def isZero () = false
  def succ () = S(this)
 def pred () = v
```

The nat Package

The Scala Programming Language

Troy Hut and Benjamin Killeen

10000

Illustrative

```
package nat
trait Nat {
  def isZero: Boolean
  def pred: Nat
  def succ: Nat.
  def + (y: Nat): Nat = if (y.isZero) this else this.succ + y.pred
  def - (y: Nat): Nat = if (y.isZero) this else this.pred + y.pred
  def * (y: Nat): Nat = if (y.isZero) y else this + this * y.pred
  def ^ (y: Nat): Nat = if (y.isZero) y.succ else this * (this ^ y.pred)
  def == (y: Nat): Boolean = {
   if (this.isZero) y.isZero
    else if (v.isZero) false
    else this.pred == y.pred
  def != (v: Nat): Boolean = !(this == v)
  def > (v: Nat): Boolean = (this.isZero, v.isZero) match {
    case (true, ) => false
    case (false, true) => true
    case (false, false) => this.pred > y.pred
  def >= (y: Nat): Boolean = (this.isZero, y.isZero) match {
    case ( , true) => true
    case (true, false) => false
    case (false, false) => this.pred >= y.pred
  def <= (y: Nat): Boolean = !(this > y)
  def < (v: Nat): Boolean = !(this >= v)
```