Making Java APIs usable with Scala

Josh Cough
joshcough@gmail.com

Who Am I?

S&P Capital IQ

- Scala, Java, F#, C#, JavaScript, Ermine

Northwestern University

- NetLogo, Scala, Java, Racket, Coq

SUNY Oswego

- Java, Doug Lea

Open Source

- https://github.com/joshcough/, https://github.com/joshcough/MinecraftPlugins
- ScalaTest, SBT (a little), Scalaz (a tiny bit)

Not Me:

The Plan

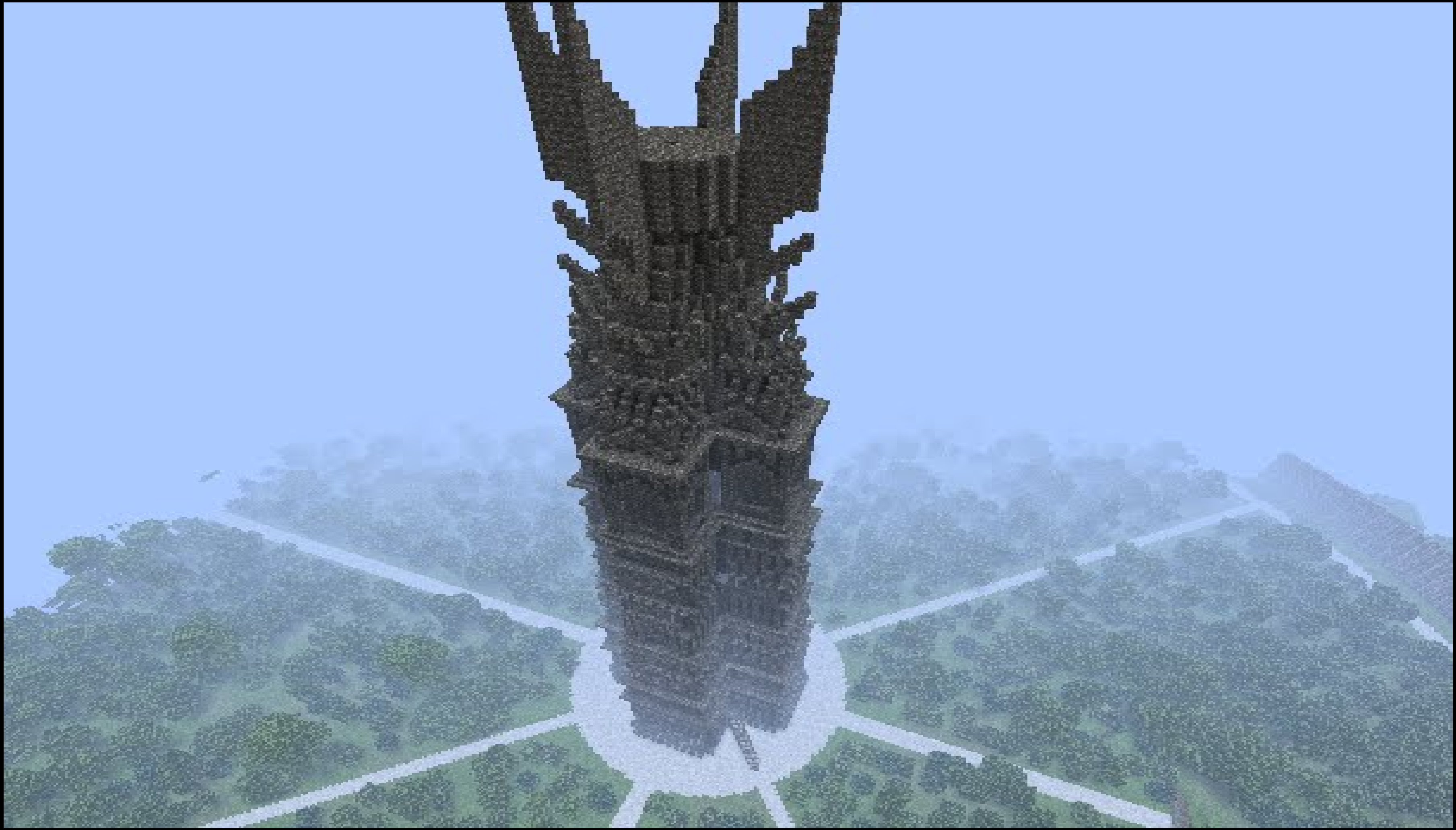
1. What is Minecraft? What is Bukkit?
2. Some Bukkit examples in Java
3. Same code redone with Scala
4. Then lots of explaining how

What is Minecraft?

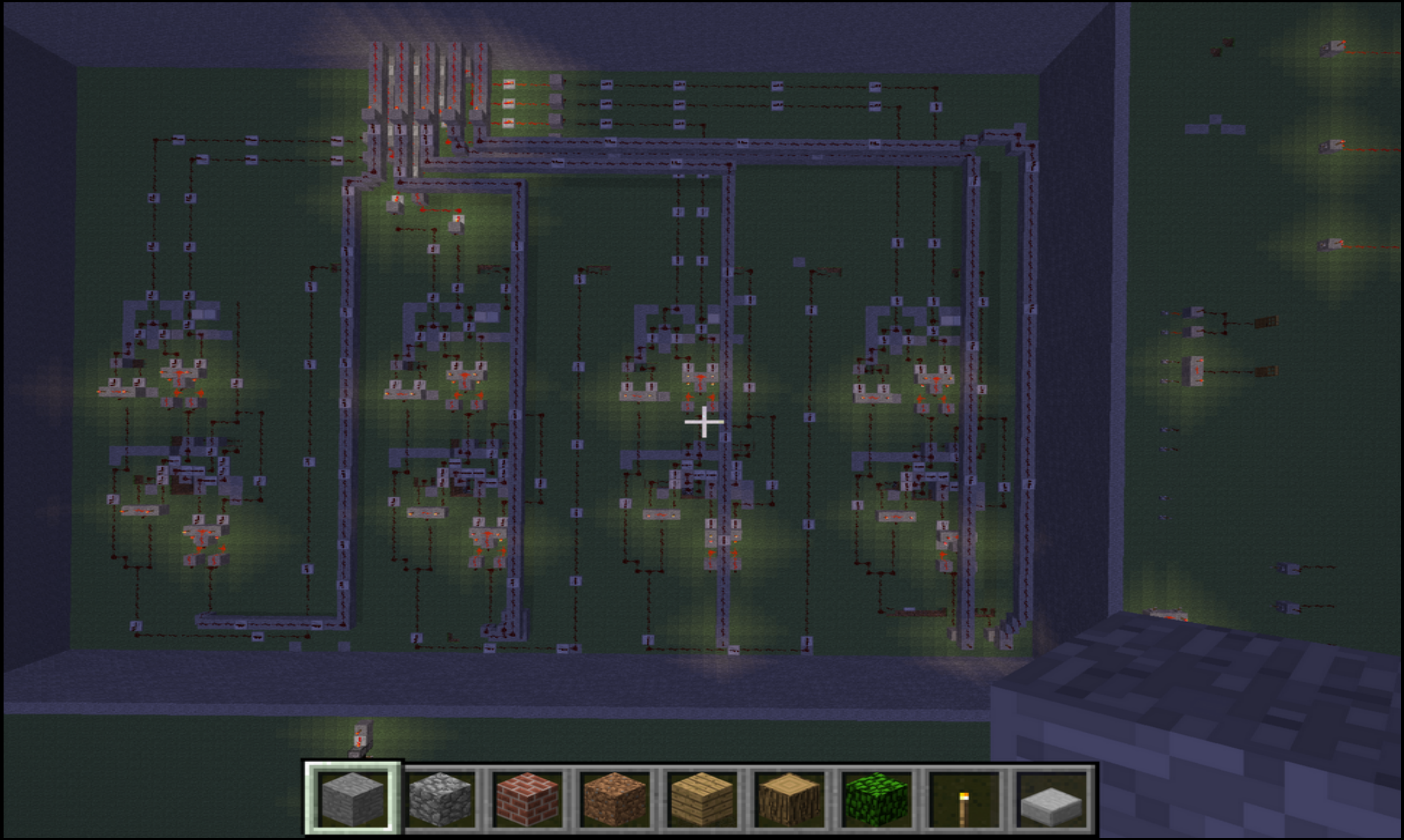
- An awesome game
- An outlet for creativity
- A real world Java project
  - Over 40 million registered users
  - At least $80 million in sales (probably a few $100m more)
- An opportunity to get people coding

Awesomeness

Creativity



Creativity



What is Bukkit?

Minecraft Server Plugin API (bukkit.org)

- Event Listeners

  An API to respond to all sorts of events that happen on the server.

- Commands

  An API to handle user commands.

- A bunch more stuff, but out of scope

Bukkit Listeners

```
// 1:  Extend JavaPlugin
public class BlockChangerGold extends JavaPlugin {

  // 2:  Create a handler for the event
  class BlockChangerListener implements Listener {
    @EventHandler
    public void onBlockDamage(BlockDamageEvent event){
      event.getBlock().setType(Material.GOLD_BLOCK);
    }
  }

  public void onEnable() {
    // 3:  Register the handler for the event
    registerEvents(new BlockChangerListener(), this);
  }
}
```

Scala Time!

Features

Things we'll definitely cover

- Higher Order Functions
- "Enrichment Classes" (new in 2.10*)
- String Interpolation (also new in 2.10)
- Parser Combinators
- For Comprehensions **
- Maybe more, time permitting.

Bukkit Listener in Scala

```scala
class BlockChangerGold extends ListeningFor(
  OnLeftClickBlock(
    (player, event) => event.block changeTo GOLD_BLOCK
))
```

Bukkit Listener in Scala

```scala
class BlockChangerGold extends ListeningFor(
  OnLeftClickBlock((_, e) => e.block changeTo GOLD_BLOCK)
)
```

Oh wait...that's better.

```
class BlockChangerGold extends ListeningFor(
  OnTouch((p, e) =>
    if (p is "joshcough") e.block changeTo GOLD_BLOCK
))
```

How did we get here?

- Higher Order Functions
- Function Literals
- Enrichment Classes

Higher Order Functions

Assertion #1: A Listener is just a Function.

Assertion #2: Functions are just Objects.

```
class MyListener implements Listener {
    public void onBlockDamage(BlockDamageEvent event)
}
```

```
class MyListener extends Function1<BlockDamageEvent, Void>{
    public void apply(BlockDamageEvent event)
}
```

Function Literals

Assertion #3: Functions are easier to create.

```
// The hard way to create a Function object
new Function1[PlayerInteractEvent, Unit]{
  def apply(e:PlayerInteractEvent): Unit = println(e)
}
```

```
// The easy way
(e:PlayerInteractEvent) => println(e)
```

```
// And in the right context, an even easier way
e => println(e)
```

HOFs vs Listeners: Quick Review

```
class BlockChangerGold extends ListeningFor(
  OnLeftClickBlock(
    (p, e) =>
      if (p is "joshcough") e.block changeTo GOLD_BLOCK
  )
)
```

HOFs vs Listeners: Quick Review

```
class BlockChangerGold extends ListeningFor(
  OnLeftClickBlock(
    (p, e) =>
      if (p is "joshcough") e.block changeTo GOLD_BLOCK
  )
)
```

```
def OnLeftClickBlock(f: (Player, PlayerInteractEvent) => Unit) =
  new Listener {
    @EventHandler
    def on(e:PlayerInteractEvent) =
      if (e.getAction == LEFT_CLICK_BLOCK) f(e.getPlayer, e)
  }
```

HOFs vs Listeners: Quick Review

```
class BlockChangerGold extends ListeningFor(
  OnLeftClickBlock(
    (p, e) =>
      if (p is "joshcough") e.block changeTo GOLD_BLOCK
  )
)
```

HOFs vs Listeners: Quick Review

```
class BlockChangerGold extends ListeningFor(
  OnLeftClickBlock(
    (p, e) =>
      if (p is "joshcough") e.block changeTo GOLD_BLOCK
  )
)
```

```
(Player, PlayerInteractEvent) => Unit
```

```
Function2[Player, PlayerInteractEvent, Unit]
```

Enrichment Classes (Scala 2.10*)

```scala
implicit class RichPlayer(player:Player){
  def is(name: String) = player.getName == name
  def ! (s:String) = if(s != null) player.sendMessage(s)
  def shock = world strikeLightning player.getLocaction
}
```

```scala
if(player is "joshcough"){
  player.shock
  player ! "zap!"
}
```

Enrichment Classes (Scala 2.10*)

```
implicit class RichBlock(b:Block) {
  def changeTo(m:Material) = b setType m
  def isNot   (m:Material) = b.getType != m
}
```

```
implicit class RichPlayerInteractEvent(e:PlayerInteractEvent){
  def block = e.getClickedBlock
}
```

```
if(e.getClickedBlock.getType != AIR)
  e.getClickedBlock setType STONE
```

```
if(e.block isNot AIR) e.block changeTo STONE
```

String Interpolation (2.10)

```
// Simple variable inside
p ! s"bc using: $m"
```

```
// Expression inside
p ! s"${p.name}, bc using: $m"
```

```
// Nested is ok too
p ! s"Awesome! ${p.name + s", bc using: $m"}"
```

```
// Escaping as you'd expect (and required here).
p ! s"You have \$500"
```

Back to Java

:(

But only for a tiny bit

BlockChanger + Commands

Let's change BlockChanger to:

- Allow users to input what Material they want to change blocks to when they punch
- Allow users to turn it on and off

Examples:

- `/bc stone`
- `/bc gold_block`
- `/bc`   `*(this turns it off)`

Quiz

What should happen if someone types these?

```
/bc gold_block

/bc

/bc eoriiweroijweorijwe

/bc dirt 7
```

Quiz

What should happen if someone types these?

`/bc gold_block`	**GOOD!**
`/bc`	**GOOD!**
`/bc eoriiweroijweorijwe`	**ERROR!**
`/bc dirt 7`	**ERROR!**

Bukkit Commands

```
// 1: still have to extend JavaPlugin
public class JavaPlugin {

    // 2: then implement this function
    //    to handle ALL of your commands
    public boolean onCommand(
        Player sender,
        String command,
        String[] args)
}
```

Disclaimer: This isn't exactly the API, but it's close enough for our purposes.

BlockChanger Revisited

Too big for slide!

BlockChanger.java

Problems with Commands

What's wrong with this API?

- A lot!
- And at least 37 other problems

Parser Combinators

Assertion: Parser Combinators are the solution to *all 42* of these problems.

Best to explain by example.

```
run(int, "5")           ⇒ Success(5,List())
run(int, "ewrer")       ⇒ Failure(invalid number: ewrer)
run(int, "7 8")         ⇒ Success(7,List(8))
run(int ~ int, "7 8")   ⇒ Success((7 ~ 8),List())
run(int ~ int, "5 qweqwe")  ⇒ Failure(invalid number: qweqwe)
run(int ~ anyString, "5 qweqwe")  ⇒ Success((5 ~ qweqwe),List())
run(int ~ anyString, "5 qweqwe wf")  ⇒
  Success((5 ~ qweqwe),List(wf))
```

Lots of examples

```
run(bool or int , "true")    ⇒ Success(Left(true),List())
run(bool or int , "7")       ⇒ Success(Right(7),List())
run(bool or int , "qweqw")  ⇒
  Failure(invalid boolean: qweqw or invalid number: qweqw)
// implicit conversion from string to Parser[String] here.
run("test", "test")     ⇒ Success(test,List())
run("test", "er")       ⇒ Failure(expected: test, but got: er)
run(int.*, "5 7 8 9")  ⇒ Success(List(5, 7, 8, 9),List())
run(bool.+, "true false true")  ⇒
  Success(List(true, false, true),List())
run(int ^^ (x => x * x), "7")  ⇒ Success(49,List())
run(int ~ "*" ~ int ^^^ "hi!", "6 * 9")  ⇒ Success(hi!,List())
run(int.? ~ "hi", "hi")      ⇒ Success((None ~ hi),List())
run(int.? ~ "hi", "6 hi")  ⇒ Success((Some(6) ~ hi),List())
```

Minecraft Parsers

```
val gamemode: Parser[GameMode] =
  ("c" | "creative" | "1") ^^^ CREATIVE |
  ("s" | "survival" | "0") ^^^ SURVIVAL

// just the types for these to save time
val material: Parser[Material]
val player  : Parser[Player]
val location: Parser[World => Location]
```

Bukkit Commands in Scala

```
class BlockChanger extends ListenerPlugin with CommandPlugin {
  val users    = collection.mutable.Map[Player, Material]()
  val listener = OnLeftClickBlock((p, e) =>
    users.get(p).foreach(e.block changeTo _)
  )
  val command  = Command(
    name = "bc",
    desc = "Specify which material to change blocks to.",
    body = args(material.?){
      case (p,Some(m)) => users+=(p -> m); p ! s"bc using: $m"
      case (p,None) => users -= p; p ! "bc has been disabled"
    }
  )
}
```

Bukkit Commands in Scala

```
class BlockChanger extends ListenerPlugin with CommandPlugin {
  val users    = collection.mutable.Map[Player, Material]()
  val listener = OnLeftClickBlock((p, e) =>
    users.get(p).foreach(e.block changeTo _)
  )
  val command  = Command(
    name = "bc",
    desc = "Specify which material to change blocks to.",
    body = args(material.?){
      case (p,Some(m)) => users+=(p -> m); p ! s"bc using: $m"
      case (p,None) => users -= p; p ! "bc has been disabled"
    }
  )
}
```

Quiz Revisited

What should happen if someone types these?

```
/bc gold_block
run(material.?,"gold_block") ⇒ Success(Some(GOLD_BLOCK),List())
/bc
run(material.?, "")          ⇒ Success(None,List())
/bc wth
run(material.?, "wth")       ⇒ Success(None,List(wth))
/bc dirt 7
run(material.?, "dirt 7")    ⇒ Success(Some(DIRT),List(7))
```

A few more Commands

```
Command("goto", "Teleport!", args(player or location){
  case (you, Left(them)) => you.teleportTo(them)
  case (you, Right(loc)) => you.teleport(loc(you.world))
})
```

```
Command("set-time", "Sets the time.", args(int){
  case (p, n) => p.world.setTime(n)
})
```

```
Command("gm", "Set your game mode.", args(gamemode){
  case (p, gm) => p.setGameMode(gm)
})
```

Putting it all together

WorldEditDemo.scala

And some for comprehensions too...

Time Permitting

More Fun Listeners

```
class LightningArrows extends ListeningFor (
  OnEntityDamageByEntity { e =>
    if (e.damager isAn ARROW) e.damagee.shock
})
```

```
class YellowBrickRoad extends ListeningFor (
  OnPlayerMove((p, e) =>
    if (p.blockOn isNot AIR) p.blockOn changeTo GOLD_BLOCK)
)
```

```
class NoRain extends ListenerPlugin {
  val listener = OnWeatherChange(e =>
    e.cancelIf(e.rain, broadcast("Put up an umbrella.")))
}
```

Named Arguments

```
case class Player(
  name: String,
  age: Int,
  awesomeness: Int)
```

```
Player("joshcough", 33, 9999)
```

```
Player(name = "joshcough", age = 33, awesomeness = 9999)
```

```
Player("edkmett", 60, -100)
```

```
Player(age = 60, awesomeness = -100, name = "edkmett")
```

Default Arguments

```scala
case class Player(
  name: String,
  age: Int,
  awesomeness: Int = 0)
```

```scala
Player(name = "joshcough", age = 33, awesomeness = 9999)
```

```scala
Player(name = "sethtisue", age = 50)
```