Scala

The learning curve

Aleksandar Prokopec

Sometime back in 2008…

beans

Sometime back in 2008…

```
List<AbstractSingletonProxyFactoryBean> beans =
```

Sometime back in 2008…

```
List<AbstractSingletonProxyFactoryBean> beans =
  new ArrayList<AbstractSingletonProxyFactoryBean>();
```

Sometime back in 2008…

```
List<AbstractSingletonProxyFactoryBean> beans =
  new ArrayList<AbstractSingletonProxyFactoryBean>();
beans.add(myBean);
```

Sometime back in 2008…

```
List<AbstractSingletonProxyFactoryBean> beans =
  new ArrayList<AbstractSingletonProxyFactoryBean>();
beans.add(myBean);
for (b : oldBeans) {

}
```

Sometime back in 2008…

```java
List<AbstractSingletonProxyFactoryBean> beans =
  new ArrayList<AbstractSingletonProxyFactoryBean>();
beans.add(myBean);
for (b : oldBeans) {
  beans.add(modernizeBean(b));
}
```

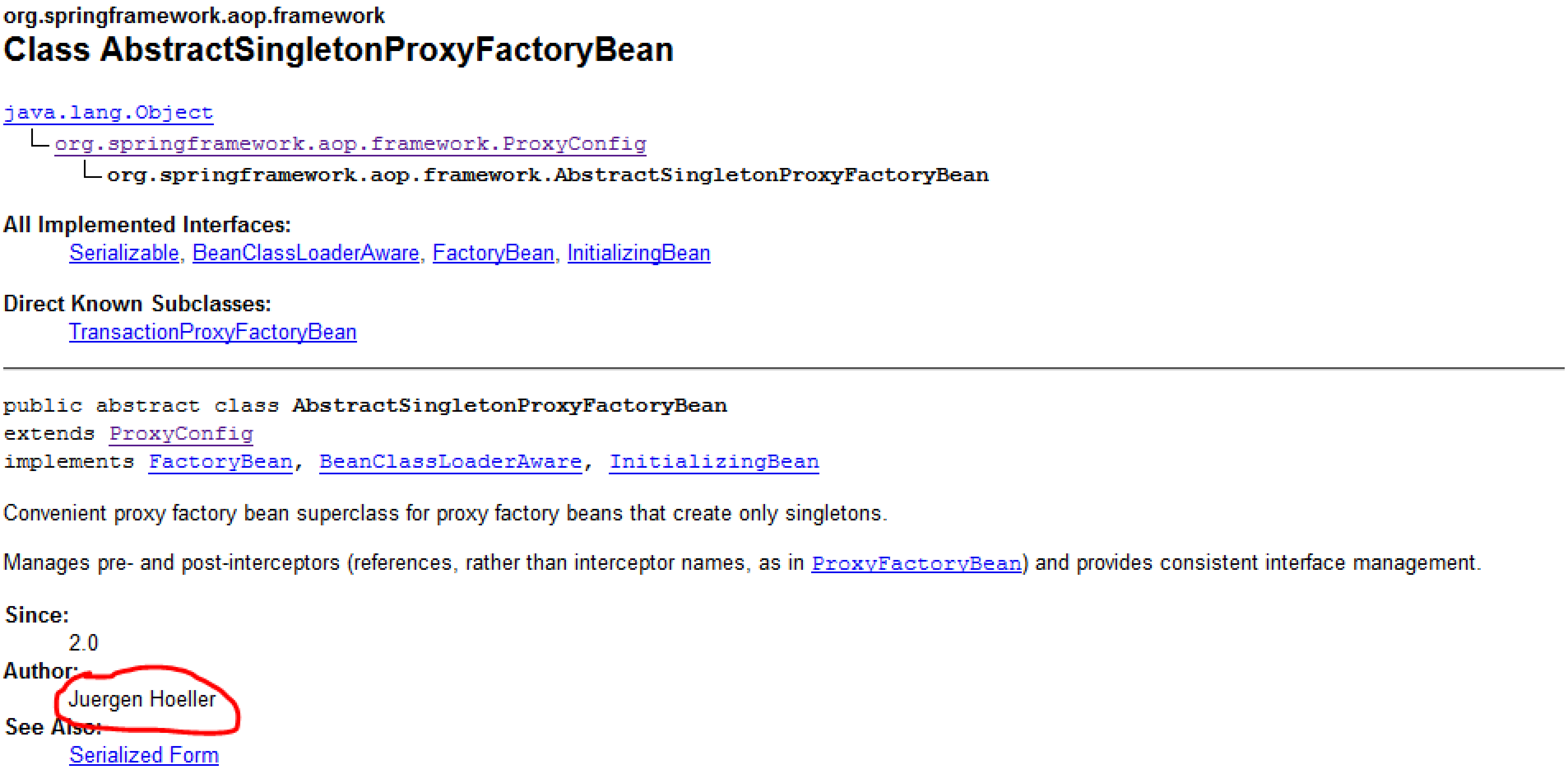
Sometime back in 2008…

```
List<AbstractSingletonProxyFactoryBean> beans =
  new ArrayList<AbstractSingletonProxyFactoryBean>();
beans.add(myBean);
for (b : oldBeans) {
  beans.add(modernizeBean(b));
}
singletonBeanMap.put(“myBeanKey”, beans);
```

Sometime back in 2008…

```
Map<
  String,
  List<AbstractSingletonProxyFactoryBean>>
  singletonBeanMap = new HashMap<
    String,
    List<AbstractSingletonProxyFactoryBean>>();

List<AbstractSingletonProxyFactoryBean> beans =
  new ArrayList<AbstractSingletonProxyFactoryBean>();
beans.add(myBean);
for (b : oldBeans) {
  beans.add(modernizeBean(b));
}
singletonBeanMap.put(“myBeanKey”, beans);
```

Advantage Pro
MPC/USB
KINESIS
Esc
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11
F12
Back Space
Delete
Option Alt
Home
End
Ctrl
Page Up
Page Down
Enter
Space
Tab
Caps Lock
Shift



org.springframework.aop.framework

Class AbstractSingletonProxyFactoryBean

```
java.lang.Object
  └ org.springframework.aop.framework.ProxyConfig
      └ org.springframework.aop.framework.AbstractSingletonProxyFactoryBean
```

All Implemented Interfaces:

Serializable, BeanClassLoaderAware, FactoryBean, InitializingBean

Direct Known Subclasses:

TransactionProxyFactoryBean

```
public abstract class AbstractSingletonProxyFactoryBean
extends ProxyConfig
implements FactoryBean, BeanClassLoaderAware, InitializingBean
```

Convenient proxy factory bean superclass for proxy factory beans that create only singletons.

Manages pre- and post-interceptors (references, rather than interceptor names, as in `ProxyFactoryBean`) and provides consistent interface management.

Since:

2.0

Author:

Juergen Hoeller

See Also:

Serialized Form

5 golden features

…that got me hooked

JVM

```
List<AbstractSingletonProxyFactoryBean> beans =
  new ArrayList<AbstractSingletonProxyFactoryBean>();
beans.add(myBean);
for (b : oldBeans) {
  beans.add(modernizeBean(b));
}
```

No semicolon

```
List<AbstractSingletonProxyFactoryBean> beans =
  new ArrayList<AbstractSingletonProxyFactoryBean>()
beans.add(myBean)
for (b : oldBeans) {
  beans.add(modernizeBean(b))
}
```

Local type inference

```
val beans =
  new ArrayList<AbstractSingletonProxyFactoryBean>()
beans.add(myBean)
for (b : oldBeans) {
  beans.add(modernizeBean(b))
}
```

Local type inference

```
val beans =
  new ArrayList[AbstractSingletonProxyFactoryBean]()
beans.add(myBean)
for (b : oldBeans) {
  beans.add(modernizeBean(b))
}
```

Collections

```
val beans =
  Buffer[AbstractSingletonProxyFactoryBean]()
beans.add(myBean)
for (b : oldBeans) {
  beans.add(modernizeBean(b))
}
```

Collections

```
val beans =
  Buffer[AbstractSingletonProxyFactoryBean]()
beans += myBean
for (b : oldBeans) {
  beans += modernizeBean(b)
}
```

Collections

```
val beans =
  Buffer[AbstractSingletonProxyFactoryBean]()
beans += myBean
for (b <- oldBeans) {
  beans += modernizeBean(b)
}
```

Collections

```
val beans =
  Buffer[AbstractSingletonProxyFactoryBean](myBean)
for (b <- oldBeans) {
  beans += modernizeBean(b)
}
```

Collections

```
val beans =
  Buffer(myBean)
for (b <- oldBeans) {
  beans += modernizeBean(b)
}
```

Collections

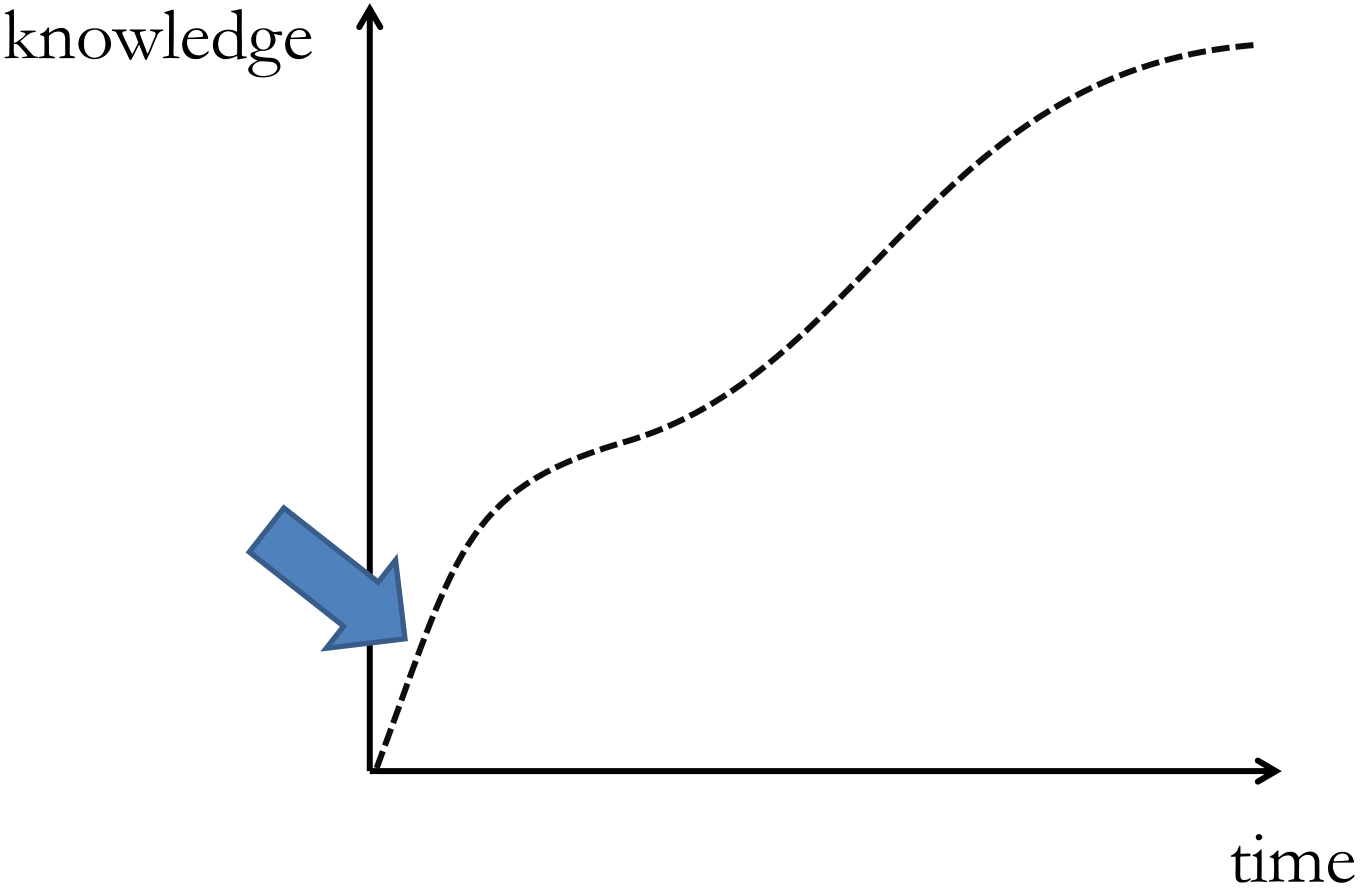
```
val beans = Buffer(myBean)
for (b <- oldBeans) {
  beans += modernizeBean(b)
}
```

Lambdas

```
val beans = Buffer(myBean)
beans ++= oldBeans.map(modernizeBean)
```

Lambdas

```
val beans = Buffer(myBean)
beans ++= oldBeans.map(modernizeBean)
```


knowledge
time

Best way to learn a language

Solve a problem

Best way to learn a language

Solve a problem

Implement a user input API
for a command-line client

Best way to learn a language

Solve a problem

Implement a user input API
for a command-line SSH client

Best way to learn a language

Solve a problem

Ask if custom host needed

If yes, ask to enter custom host

```
print(“Custom [Y/N]: “)
val yn = readln()
```

```
print(“Custom [Y/N]: “)
val yn = readln()
var host = “ ”
```

```
print(“Custom [Y/N]: “)
val yn = readln()
var host = “”
if (yn.trim == “Y”) {


} else ...
```

```
print(“Custom [Y/N]: “)
val yn = readln()
var host = “ ”
if (yn.trim == “Y”) {
  print(“Enter host: “)
  host = readln()
} else ...
```

```
print(“Custom [Y/N]: “)
val yn = readln()
var host = “”
if (yn.trim == “Y”) {
  print(“Enter host: “)
  host = readln()
} else host = “server.lan:22”
```

```
print(“Custom [Y/N]: “)
val yn = readln()
var host = “ ”
if (yn.trim == “Y”) {
  print(“Enter host: “)
  host = readln()
} else host = “server.lan:22”
connect(new Remote(host))
```

```
print(“Custom [Y/N]: “)
val yn = readln()
var host = “”
if (yn.trim == “Y”) {
  print(“Enter host: “)
  host = readln()
} else host = “server.lan:22”
connect(new Remote(host))

> Custom [Y/N]:
java.lang.NullPointerException
```

Fundamental problem

`readln()` returns `null` for empty entries

I call it my billion-dollar mistake. It was the invention of the null reference in 1965.

Tony Hoare

```
abstract class Option[T]
abstract class Option[T]

class Some[T](x: T) extends Option[T]

class None[T] extends Option[T]
```

```scala
abstract class Option[T] {
  def get: T
}

class Some[T](x: T) extends Option[T]

class None[T] extends Option[T]
```

```scala
abstract class Option[T] {
  def get: T
}

class Some[T](x: T) extends Option[T] {
  def get = x
}

class None[T] extends Option[T]
```

```scala
abstract class Option[T] {
  def get: T
}

class Some[T](x: T) extends Option[T] {
  def get = x
}

class None[T] extends Option[T] {
  def get = sys.error(“None.get”)
}
```

```
def text(q: String): Option[String] = {
  print(q)
  val input = readln()
  if (input != null) Some(input)
  else None
}
```

```
val yn = text(“Custom [Y/N]: “)
val yn = text(“Custom [Y/N]: “)
var host = “ ”
if (yn.trim == “Y”) {
  host = text(“Enter host: “)
}
```

```
val yn = text(“Custom [Y/N]: “)
var host = “”
if (yn.trim == “Y”) {
  host = text(“Enter host: “)
} else host = “server.lan:22”
connect(new Remote(host))
```

```
val yn = text(“Custom [Y/N]: “)
var host = “ ”
if (yn.trim == “Y”) {
  host = text(“Enter host: “)
} else host = “server.lan:22”
connect(new Remote(host))

error: trim not a member of Option[String]
              yn.trim
                 ^
```

```
val yn: Option[String] = text(“Custom: “)
var host = “”
if (yn.trim == “Y”) {
  host = text(“Enter host: “)
} else host = “server.lan:22”
connect(new Remote(host))

error: trim not a member of Option[String]
              yn.trim
                 ^
```

```
val yn = text(“Custom [Y/N]: “).get
var host = “”
if (yn.trim == “Y”) {
  host = text(“Enter host: “).get
} else host = “server.lan:22”
connect(new Remote(host))
```

```
val yn = text(“Custom [Y/N]: “).get
var host = “”
if (yn.trim == “Y”) {
  host = text(“Enter host: “).get
} else host = “server.lan:22”
connect(new Remote(host))

> Custom [Y/N]:
java.lang.RuntimeError: None.get
```

```
val yn = text(“Custom [Y/N]: “)
if (yn == None) return
var host: Option[String] = None
if (yn.get.trim == “Y”) {
  val h = text(“Enter host: “)
  if (h == None) return else host = h
} else host = Some(“server.lan:22”)
connect(new Remote(host.get))
```

```
val yn = text(“Custom [Y/N]: “)
if (yn == None) return
var host: Option[String] = None
if (yn.get.trim == “Y”) {
  val h = text(“Enter host: “)
  if (h == None) return else host = h
} else host = Some(“server.lan:22”)
return new Remote(host.get)

def startClient() {
  val remote: Option[Remote] = query()
  connect(remote)
}
```

```
def query(): Option[Remote] = {
  val yn = text(“Custom [Y/N]: “)
  if (yn == None) return None
  var host: Option[String] = None
  if (yn.get.trim == “Y”) {
    val h = text(“Enter host: “)
    if (h == None) return None
    else host = h
  } else host = Some(“server.lan:22”)
  return Some(new Remote(host.get))
}

def startClient() {
  val remote: Option[Remote] = query()
  connect(remote)
}
```

```
def query(): Option[Remote] = {
  val yn = text(“Custom [Y/N]: “)
  if (yn == None) return None
  var host: Option[String] = None
  if (yn.get.trim == “Y”) {
    val h = text(“Enter host: “)
    if (h == None) return None
    else host = h
  } else host = Some(“server.lan:22”)
  return Some(new Remote(host.get))
}

def startClient() {
  val remote: Option[Remote] = query()
  connect(remote)
}
```

```
def startClient() {
  println(“Now selecting remote.“)
  val remote: Option[Remote] = query()
  remote match {


  }
}
```

```
def startClient() {
  println(“Now selecting remote.“)
  val remote: Option[Remote] = query()
  remote match {
    case Some(r) => connect(r)

  }
}
```

```
def startClient() {
 println(“Now selecting remote.“)
 val remote: Option[Remote] = query()
 remote match {
  case Some(r) => connect(r)
  case None => println(“Can’t connect.”)
 }
}
```

```
def startClient() {
  println(“Now selecting remote.“)
  val remote: Option[Remote] = query()
  remote match {
    case Some(r) => connect(r)
    case None => println(“Can’t connect.”)
  }
}

if (flag_testing)
  startClient(() => Some(localhost))
else
  startClient(query)
```

```
def startClient(q: () => Option[Remote]) {
  println(“Now selecting remote.“)
  val remote: Option[Remote] = q()
  remote match {
    case Some(r) => connect(r)
    case None => println(“Can’t connect.”)
  }
}

if (flag_testing)
  startClient(() => Some(localhost))
else
  startClient(query)
```

```
def query: () => Option[Remote] = {
  () =>
  val yn = text(“Custom [Y/N]: “)
  if (yn == None) return None
  var host: Option[String] = None
  if (yn.get.trim == “Y”) {
    val h = text(“Enter host: “)
    if (h == None) return None
    else host = h
  } else host = Some(“server.lan:22”)
  return Some(new Remote(host.get))
}
```

Token soup

Token soup

An incomprehensible jumble of characters which it is difficult or impossible to discern the meaning from.

No artificial colours or flavours
Aucun colorant ni saveur artificiels
Grandma's
SOUPE CONDENSÉE
TOKEN SOUP
100% HOMEMADE
HOMESTYLE CHICKEN NOODLE
POULET ET NOUILLES GRAND-MÈRE

Plan to throw one away; you will, anyhow.

Fred Brooks
The Mythical Man-Month

```
() => Option[T]
```

```
type Query[T] = () => Option[T]
type Query[T] = () => Option[T]

def const[T](x: T): Query[T] =
type Query[T] = () => Option[T]

def const[T](x: T): Query[T] =
  () => Some(x)
```

```scala
type Query[T] = () => Option[T]

def const[T](x: T): Query[T] =
  () => Some(x)

startClient(
  const(new Remote(“localhost”)))
```

```
type Query[T] = () => Option[T]

def text(q: String): Query[String] = {



}
```

```scala
type Query[T] = () => Option[T]

def text(q: String): Query[String] = {
  () =>




}
```

```
type Query[T] = () => Option[T]

def text(q: String): Query[String] = {
  () =>
  print(q)



}
```

```scala
type Query[T] = () => Option[T]

def text(q: String): Query[String] = {
  () =>
  print(q)
  val input = readln()


}
```

```
type Query[T] = () => Option[T]

def text(q: String): Query[String] = {
  () =>
  print(q)
  val input = readln()
  if (input != null) Some(input)
  else None
}
```

```
val hostQuery = text(“Enter host: “)
val hostQuery: Query[String] =
  text(“Enter host: “)
```

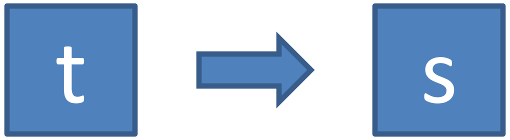
```
val hostQuery: Query[String] =
  text(“Enter host: “)

val remoteQuery: Query[Remote]
val hostQuery: Query[String] =
  text(“Enter host: “)

val remoteQuery: Query[Remote] =
                host => new Remote(host)
```

```
val hostQuery: Query[String] =
  text(“Enter host: “)

val remoteQuery: Query[Remote] =
  hostQuery.map(host => new Remote(host))
```

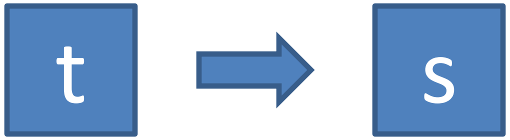
```
val hostQuery: Query[String] =
  text(“Enter host: “)

val hostOption: Option[String] =
  hostQuery()

val remoteOption: Option[Remote] =
  hostOption.map(host => new Remote(host))
```

```
val hostQuery: Query[String] =
  text(“Enter host: “)

val hostOption: Option[String] =
  hostQuery()

val remoteOption: Option[Remote] =
  hostOption.map(host => new Remote(host))

  for (host <- hostOption) yield {
    new Remote(host)
  }
```

```
def map[T, S](q: Option[T], f: T => S):
  Option[S] =
def map[T, S](q: Option[T], f: T => S):
  Option[S] =
```



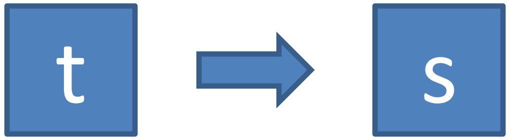
```
def map[T, S](q: Option[T], f: T => S):
  Option[S] = q match {


}
```



t

s

```
def map[T, S](q: Option[T], f: T => S):
  Option[S] = q match {
  case Some(v) => Some(f(v))
  case None => None
}
```



t

s

```scala
type Query[T] = () => Option[T]

def map[T, S]
  (q: Query[T], f: T => S): Query[S] =
type Query[T] = () => Option[T]

def map[T, S]
  (q: Query[T], f: T => S): Query[S] =
  () =>
```

```
type Query[T] = () => Option[T]

def map[T, S]
  (q: Query[T], f: T => S): Query[S] =
  () => q()
```

```
type Query[T] = () => Option[T]

def map[T, S]
  (q: Query[T], f: T => S): Query[S] =
  () => q() match {


  }
```

```
type Query[T] = () => Option[T]

def map[T, S]
  (q: Query[T], f: T => S): Query[S] =
  () => q() match {
    case Some(v) => Some(f(v))
    case None => None
  }
```

```
val hostQuery: Query[String] =
  text(“Enter host: “)

val remoteQuery: Query[String] =
  for (host <- hostQuery) yield {
    new Remote(host)
  }
```

```scala
for (host <- hostQuery) yield {
  new Remote(host)
}
```

```
val yn = text(“Custom [Y/N]: “)


for {
  host <- if (???) text(“Host: “)
          else const(“server.lan:22”)
} yield new Remote(host)
```

```
val yn: Query[String] =
  text(“Custom [Y/N]: “)

for {
  host <- if (???) text(“Host: “)
          else const(“server.lan:22”)
} yield new Remote(host)
```

```
for {
  yn   <- text(“Custom [Y/N]”)
  host <- if (???) text(“Host: “)
          else const(“server.lan:22”)
} yield new Remote(host)
```

```
for {
  yn   <- text(“Custom [Y/N]”)
  host <- if (yn == “Y”) text(“Host: “)
          else const(“server.lan:22”)
} yield new Remote(host)
```

```
for {
  yn   <- Some(“Y”)
  host <- if (yn == “Y”) Some(“localhost”)
          else Some(“server.lan:22”)
} yield new Remote(host)
```

```
for {
  yn   <- Some(“Y”)
  host <- Some(“localhost”)
} yield new Remote(host)
```

```
for {
  yn   <- Some(“Y”)
  host <- Some(“localhost”)
} yield new Remote(host)
```

```
Some(“Y”).map(yn =>



)
```

```
for {
  yn   <- Some(“Y”)
  host <- Some(“localhost”)
} yield new Remote(host)


Some(“Y”).map(yn =>
  Some(“localhost”)


)
```

```
for {
  yn   <- Some(“Y”)
  host <- Some(“localhost”)
} yield new Remote(host)
```

```
Some(“Y”).map(yn =>
  Some(“localhost”).map(host =>
    new Remote(host)
  )
)
```

```
for {
  yn   <- Some(“Y”)
  host <- Some(“localhost”)
} yield new Remote(host)
```

```
Some(“Y”).map(yn =>
  Some(“localhost”).map(host =>
    new Remote(host)
  )
): Option[???]
```

```
for {
  yn   <- Some(“Y”)
  host <- Some(“localhost”)
} yield new Remote(host)
```

```
Some(“Y”).map(yn =>
  Some(“localhost”).map(host =>
    new Remote(host)
  )
): Option[Option[Remote]]
```

```
def map[T, S](q: Option[T],
  f: T => S): Option[S]
```

```
def map[T, Option[S]](q: Option[T],
  f: T => Option[S]): Option[Option[S]]
```

```
def flatMap[T, S](q: Option[T],
  f: T => Option[S]): Option[S]
```

```
def flatMap[T, S](q: Option[T],
  f: T => Option[S]): Option[S] = {
  q match {
    case None => None

  }
}
```

```
def flatMap[T, S](q: Option[T],
  f: T => Option[S]): Option[S] = {
  q match {
    case None => None
    case Some(v) => f(v)
  }
}
```

```
def flatMap[T, S](q: Option[T],
  f: T => Option[S]): Option[S]

for {
  yn   <- Some(“Y”)
  host <- if (yn == “Y”) Some(“localhost”)
          else Some(“server.lan:22”)
} yield new Remote(host)
```

```
Some(“Y”).flatMap(yn =>
  Some(“localhost”).map(host =>
    new Remote(host)
  )
): Option[Remote]
```

```
def flatMap[T, S](q: Option[T],
  f: T => Option[S]): Option[S]
```

```
for {
  yn   <- Some(“Y”)
  host <- if (yn == “Y”) Some(“localhost”)
          else Some(“server.lan:22”)
} yield new Remote(host)
```

```
Some(“Y”).flatMap(yn =>
  Some(“localhost”).map(host =>
    new Remote(host)
  )
): Option[Remote]
```

```
def flatMap[T, S](q: Option[T],
  f: T => Option[S]): Option[S]

for {
  yn   <- Some(“Y”)
  host <- if (yn == “Y”) Some(“localhost”)
          else Some(“server.lan:22”)
} yield new Remote(host)

Some(“Y”).flatMap(yn =>
  Some(“localhost”).map(host =>
    new Remote(host)
  )
): Option[Remote]
```

```
type Query[T] = () => Option[T]

def flatMap[T, S](q: Query[T],
  f: T => Query[S]): Query[S]
```

```scala
type Query[T] = () => Option[T]

def flatMap[T, S](q: Query[T],
  f: T => Query[S]): Query[S] =
  () => q() match {
    case Some(v) => f(v)()
    case None => None
  }
```

```
for {
  yn   <- text(“Custom [Y/N]: “)
  host <- if (yn == “Y”) text(“Host: “)
          else const(“server.lan:22”)
} yield new Remote(host)
```

`Query[T]`

is a monad!

```
val yn = text(“Custom [Y/N]? ”)()
if (yn == None) return None
var host = null
if (yn.get == “Y”) host = text(“Host: ”)()
else host = const(“server.lan:22”)()
if (host == None) return None
return Some(new Remote(host.get))
```

VS.

```
for {
  yn   <- text(“Custom [Y/N]: “)
  host <- if (yn == “Y”) text(“Host: “)
          else const(“server.lan:22”)
} yield new Remote(host)
```

Argument #1

The monad approach is more concise, more readable and more beautiful.

```
val yn = text(“Custom [Y/N]? ”)()
if (yn == None) return None
var host = null
if (yn.get == “Y”) host = text(“Host: ”)()
else host = const(“server.lan:22”)()
if (host == None) return None
return Some(new Remote(host.get))
```

```
for {
  yn   <- text(“Custom [Y/N]: “)
  host <- if (yn == “Y”) text(“Host: “)
          else const(“server.lan:22”)
} yield new Remote(host)
```

```
val yn = text(“Custom [Y/N]? ”)()
if (yn == None) return None
var host = null
if (yn.get == “Y”) host = text(“Host: ”)()
else host = const(“server.lan:22”)()
if (host == None) return None
return Some(new Remote(host.get))
```

```
for {
  yn   <- text(“Custom [Y/N]: “)
  host <- if (yn == “Y”) text(“Host: “)
          else const(“server.lan:22”)
} yield new Remote(host)
```

But, beauty is in the eye of the beholder.

Argument #2

The monad approach is shorter.

```
val yn = text(“Custom [Y/N]? ”)()
if (yn == None) return None
var host = null
if (yn.get == “Y”) host = text(“Host: ”)()
else host = const(“server.lan:22”)()
if (host == None) return None
return Some(new Remote(host.get))
```

```
for {
  yn   <- text(“Custom [Y/N]: “)
  host <- if (yn == “Y”) text(“Host: “)
          else const(“server.lan:22”)
} yield new Remote(host)
```

But, does shorter code imply better code?

```
while(*dst++ = *src++);
```

Argument #3

In the monad approach,
there is no duplicated code.

```
val yn = text(“Custom [Y/N]? ”)()
if (yn == None) return None
var host = null
if (yn.get == “Y”) host = text(“Host: ”)()
else host = const(“server.lan:22”)()
if (host == None) return None
return Some(new Remote(host.get))
```

```
for {
  yn   <- text(“Custom [Y/N]: “)
  host <- if (yn == “Y”) text(“Host: “)
          else const(“server.lan:22”)
} yield new Remote(host)
```

```
val yn = text(“Custom [Y/N]? ”)()
if (yn == None) return None
var host = null
if (yn.get == “Y”) host = text(“Host: ”)()
else host = const(“server.lan:22”)()
if (host == None) return None
return Some(new Remote(host.get))
```

Not just more boilerplate,
more of the **same** code.

DRY principle

Reduce repetition of any kind.

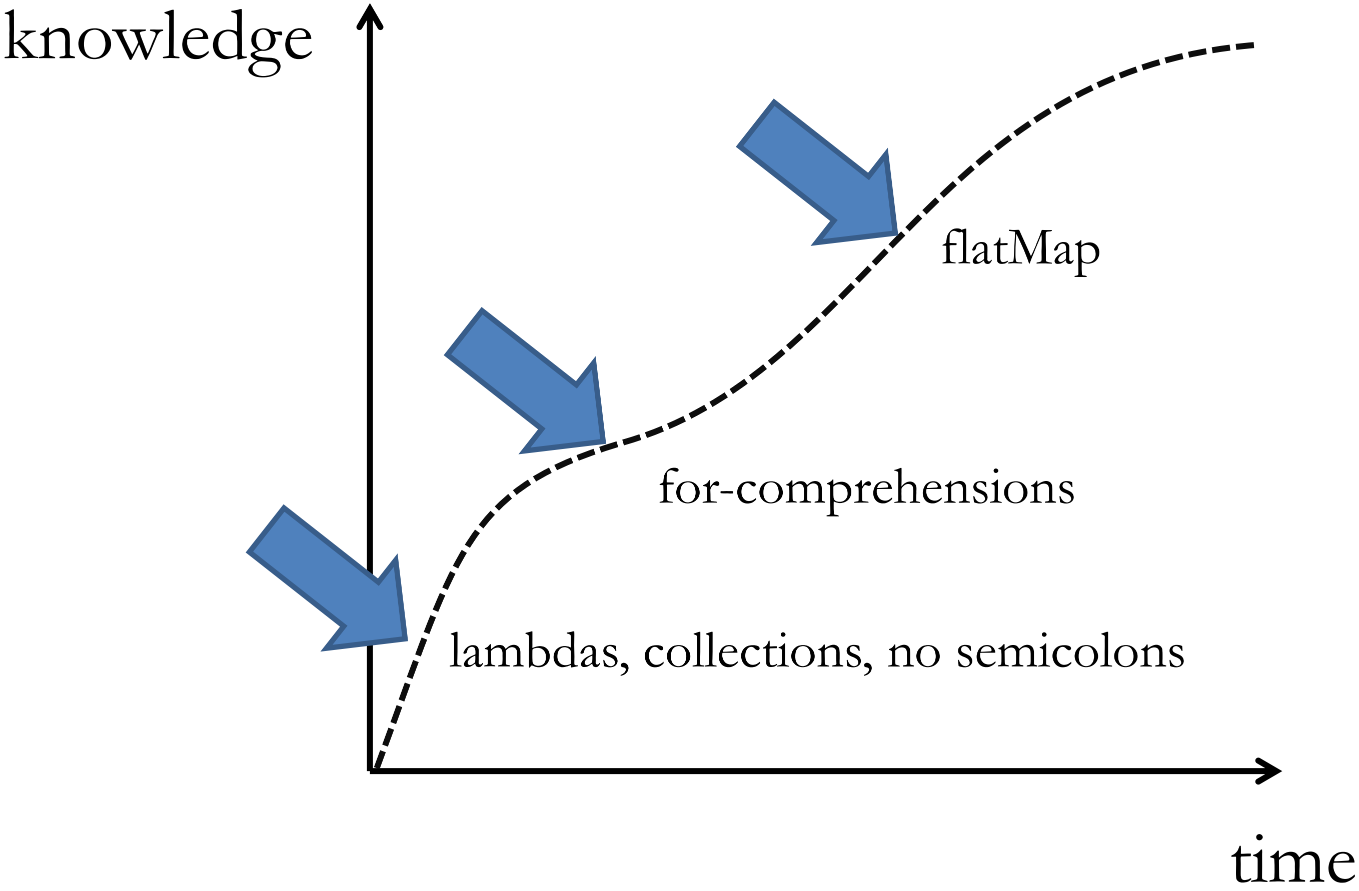
Monad

Abstracts how the statements in the program are chained together.

Monad

A programmable semicolon.

Congratulations, you understand monads!

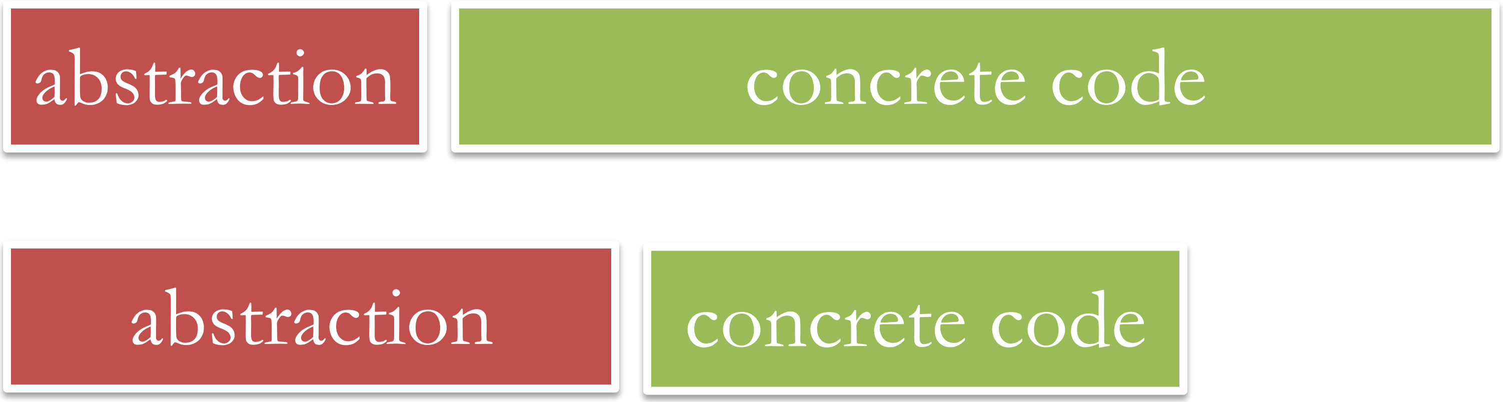


flatMap

for-comprehensions

lambdas, collections, no semicolons

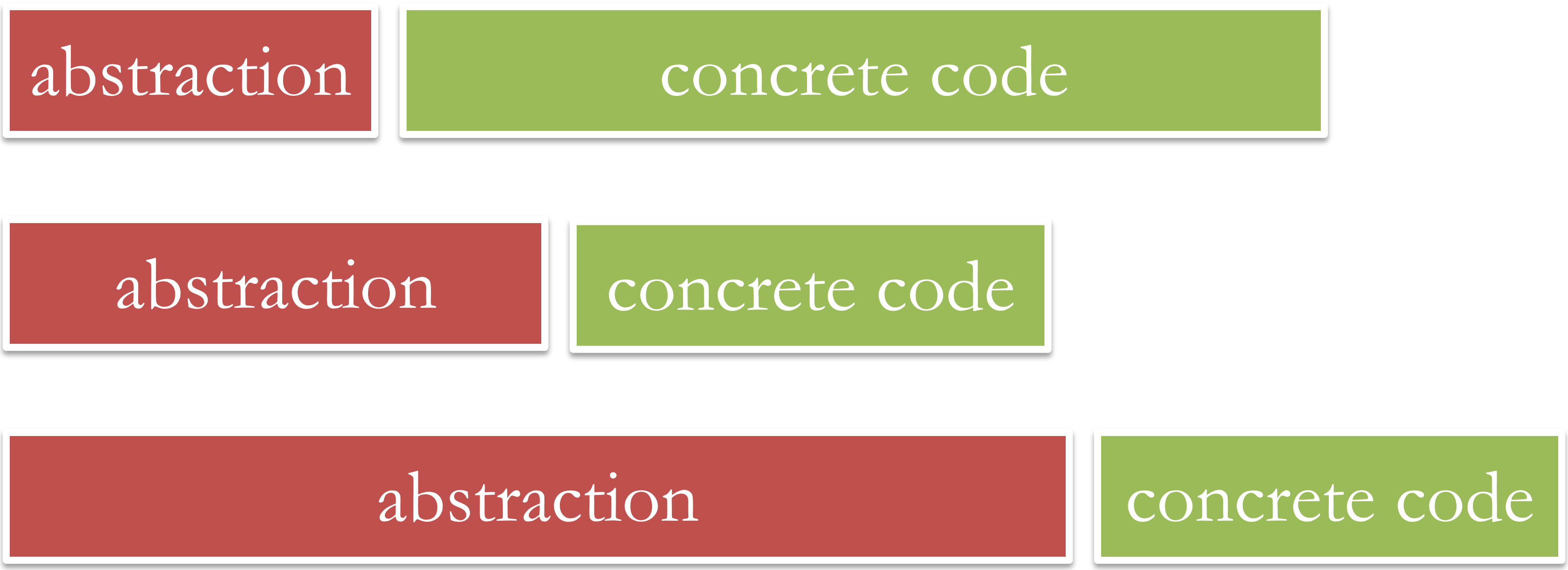
Abstraction



abstraction concrete code

abstraction concrete code

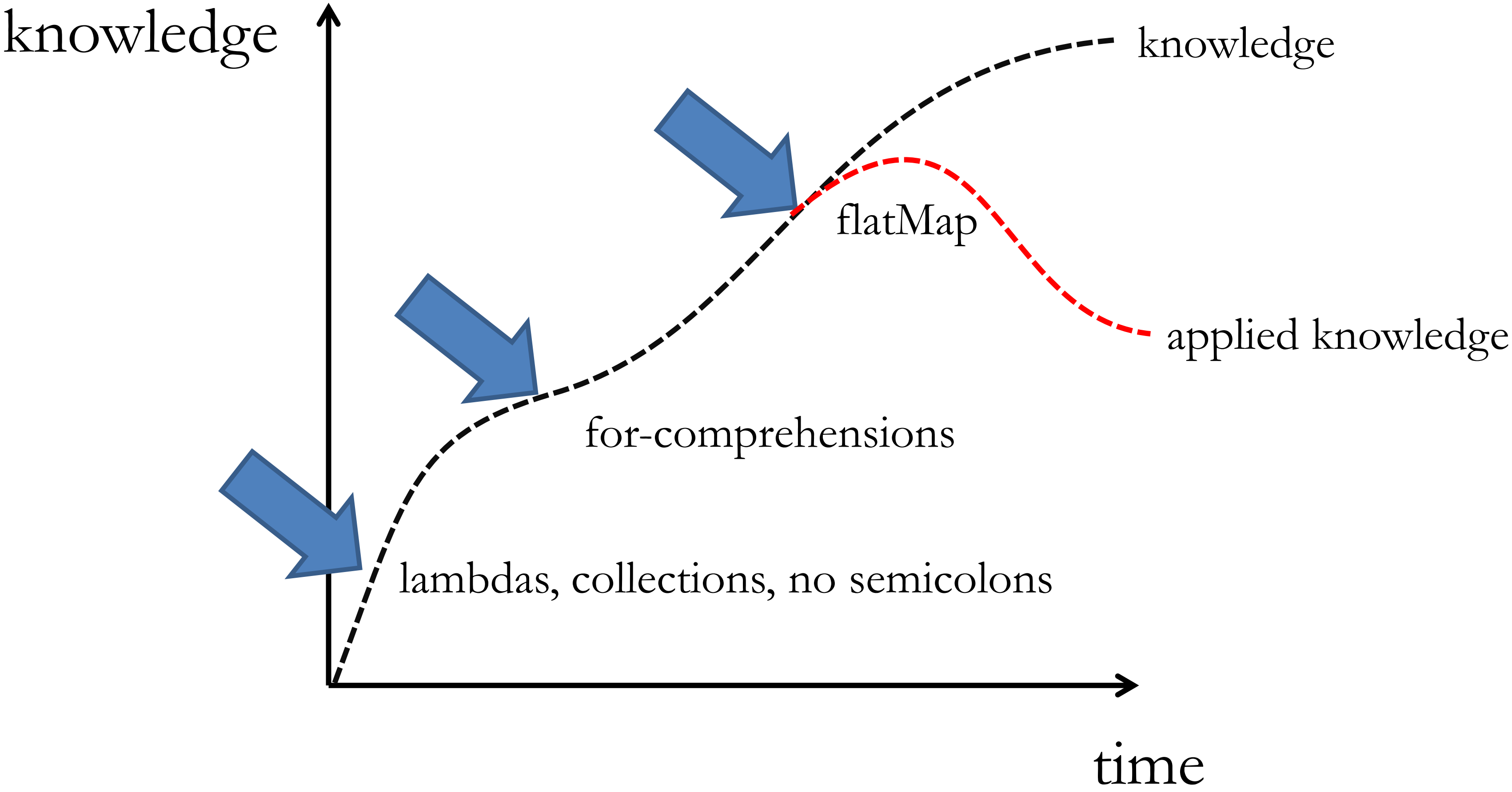
Abstraction



abstraction concrete code

abstraction concrete code

abstraction concrete code


knowledge
knowledge
flatMap
applied knowledge
for-comprehensions
lambdas, collections, no semicolons
time

Disallow abstractions

Disallow abstractions

JOB INTERVIEW
TELL ME YOUR PROCESS FOR SOLVING THIS SORT OF PROBLEM.
Dilbert.com DilbertCartoonist@gmail.com
I WOULD IGNORE IT FOR A WEEK AND LIKELY DISCOVER THAT IT WASN'T IMPORTANT IN THE FIRST PLACE.
8-1-15 © 2015 Scott Adams, Inc. /Dist. by Universal Uclick
IF IT STILL MATTERS AFTER A WEEK, I WOULD HOLD FAKE JOB INTER– VIEWS AND ASK PEOPLE HOW TO SOLVE IT.
APPARENTLY, THAT DOESN'T WORK.

Better tooling

Programmable lint checkers

It’s useful to customize your semicolons,
but even more useful to omit them.

Thank you!