Hola Amigos!!
!!
!!
!!
!!
!!
!!

[note to self] – Scream this greeting as loud as possible

This talk is about Effective Java and how Groovy makes it easy to follow the idioms from the book. It's also about new tips that would be in Effective Groovy were someone to write book.

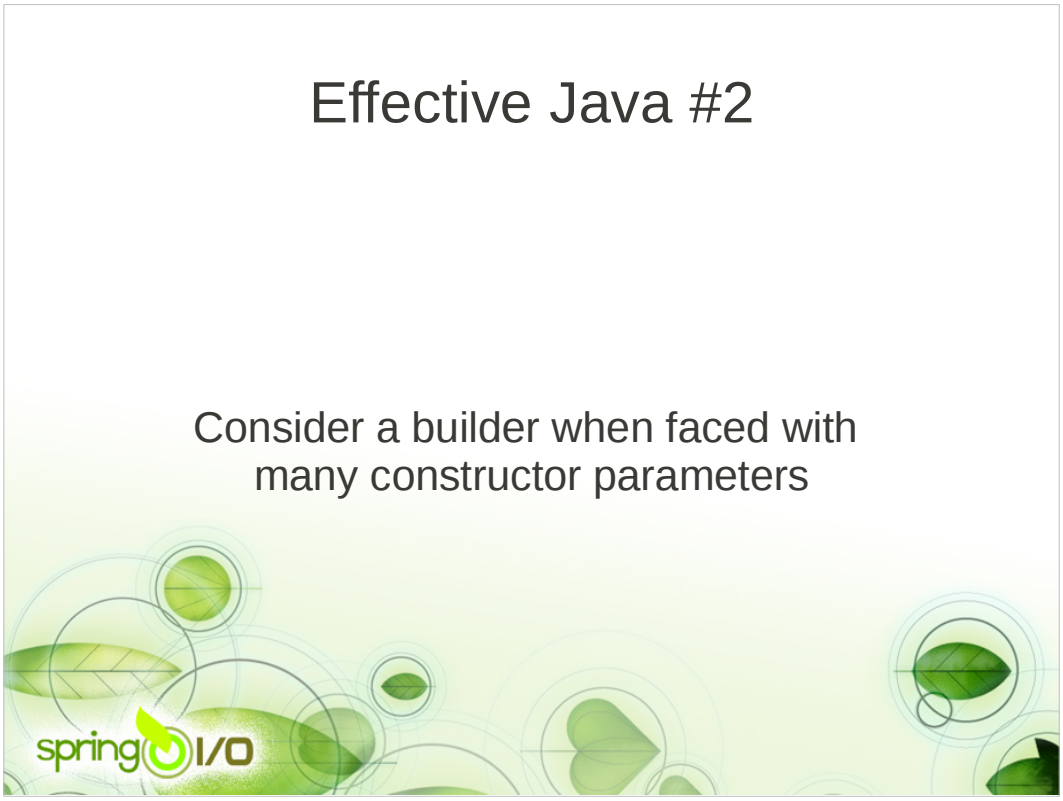
And no, I will not write that book.

About Me Reminders:

* We have JetBrains shirts at Canoo Booth
* Canoo is a JetBrains Partner
* Make that super funny Barney Miller joke

The code this this preso is up on github:
https://github.com/HamletDRC/presentations/tree/master/effectivegroovy

Hopefully mentioning GitHub will give me some credibility that I'd otherwise be lacking.



Motivating Example: Super long java constructor with a bunch of int or boolean parameters.
/src/ej02/01.groovy

Java – Write a builder object
/src/ej02/02.groovy
1) Show target syntax with*()
2) Show Java boilerplate (show immutability)

Groovy – Use @Immutable
/src/ej02/01.groovy
1) add @Immutable & remove constructor
2) run example
3) WHEEE!!!

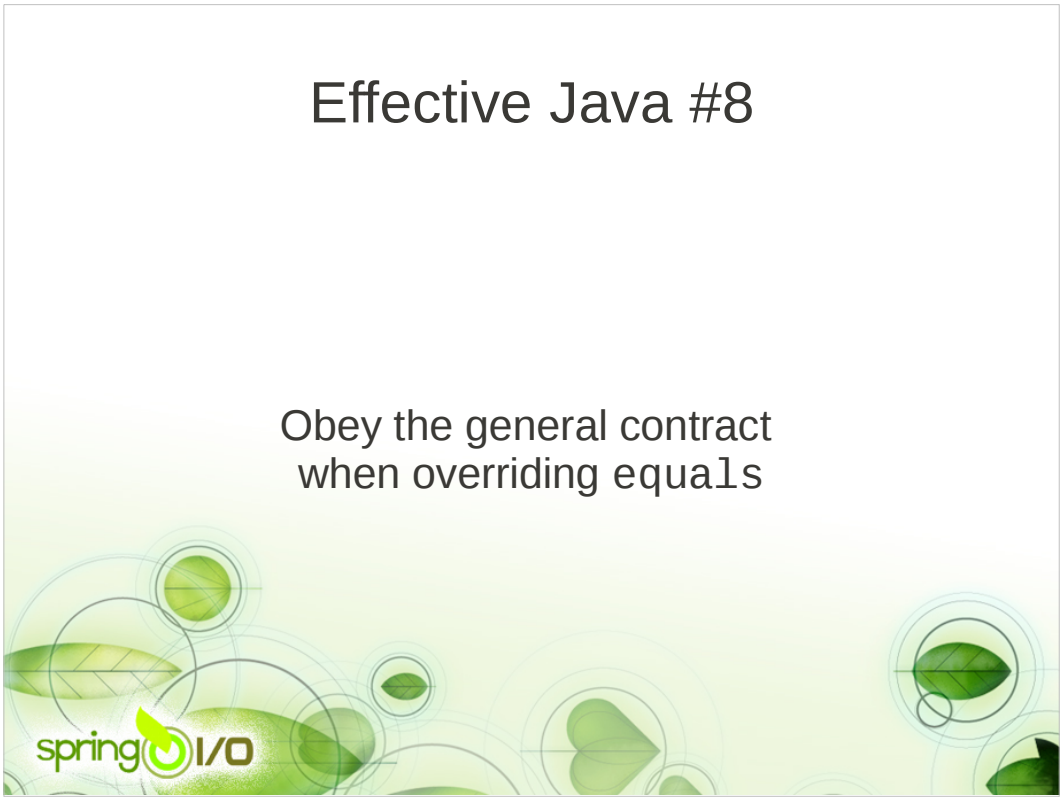
Did you notice we used @Immutable in the last slide?

That's a Java tip as well: prefer immutable objects.

CodeNarc helps you remember to use groovy.transform.Immutable and **not** groovy.lang.Immutable, which is deprecated.

src/ej05/01.groovy
Groovy has waaay more type literals than Java.
1) Do not create new instances of primitive types
2) Groovy has BigDecimal/BigInteger literals
3) Groovy has List/Map type literals, which can be used even for your own List/Map types
4) Effective Java Motivation: Don't create Calendar/Date instances unnecessarily

– both CodeNarc and IDEA catch these for you
– CodeNarc catches more of them than IDEA



src/ej08/01.groovy
Equals Must Be: Reflexive, symmetric, transitive, consistent, null is false

1) Generate equals and hashcode into Person
2) Replace with @EqualsAndHashCode

If you don't do this then hashmaps have big issues later.

CodeNarc & IDEA have analysis for this common problem

If you don't, then it's often a Pain In The Butt when there is a problem. Is it that hard?

1) Generate toString using IDE
2) Replace with @ToString

Makes Groovy a 'real' scripting language

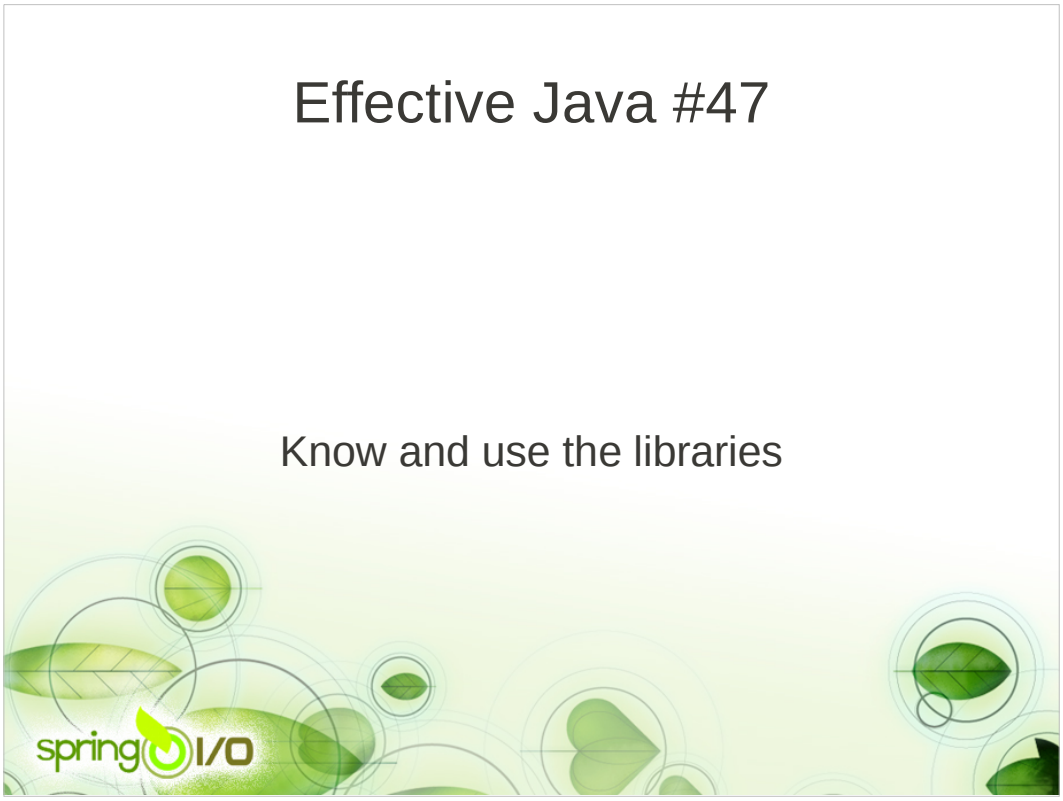
time groovy -e "println new Date()"
Wed Feb 09 06:41:32 CET 2011

real 0m0.666s
user 0m0.750s
sys 0m0.040s

time groovyclient -e "println new Date()"
Wed Feb 09 06:42:48 CET 2011

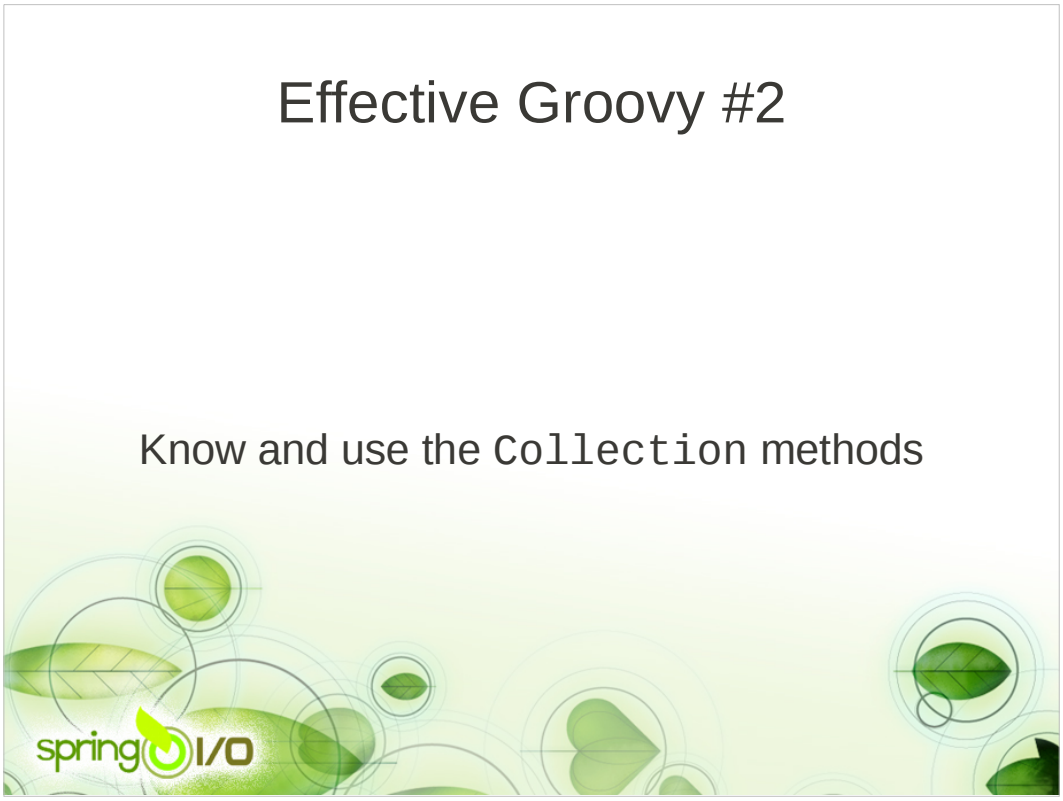
real 0m0.040s
user 0m0.010s
sys 0m0.000s

http://kobo.github.com/groovyserv/
Put in your path
alias groovy=groovyclient



Read read read.

You don't need to know the details of an API, simply that an API exists. Don't momorize the parameter types to methods, or the methods on an Object, but memorize which Objects the JDK and GDK has.



These are the bread and butter of the Effective Groovy programmer.

src/eg02/01.groovy

1) Should be .collect

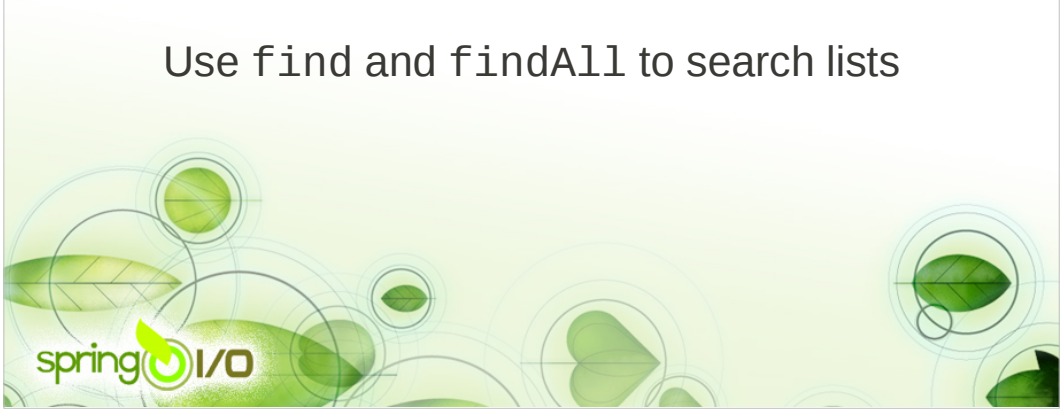
```
List<Person> people = Person.findAll();

List<Integer> ids = new ArrayList<Integer>();
for (Person p : people) {
    ids.add(p.getId());
}
```

2) Should really be *. (spread)

CodeNarc can find usages that should be * instead of .collect

Effective Groovy #2b

Use `find` and `findAll` to search lists

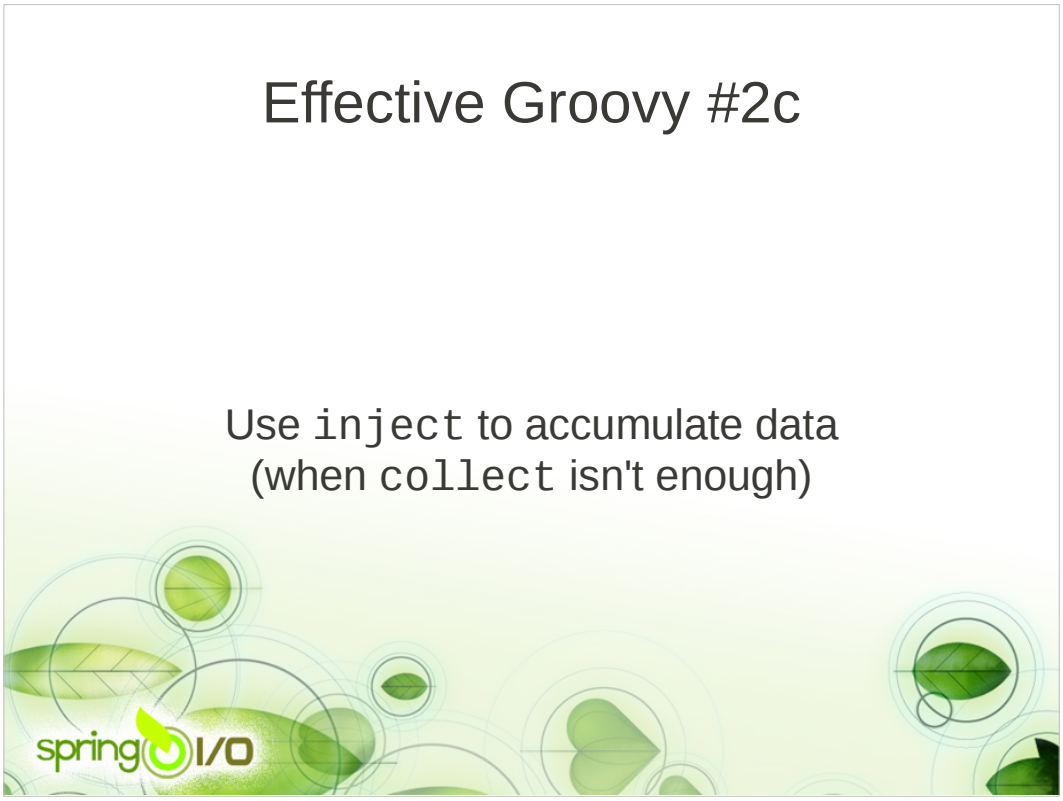


src/eg02/02.groovy
Should be .find, not iteration

```
List<Person> people = Person.findAll();
Person joe = null;
for (Person p : people) {
    if ("Joe".equals(p.getFirstName())) {
        joe = p;
        break;
    }
}
```

src/eg02/03.groovy
Should be .findAll, not iteration

```
List<Person> people = Person.findAll();

List<Person> bucks = new ArrayList<Person>();
for (Person p : people) {
    if ("Buck".equals(p.getLastName())) {
        bucks.add(p);
    }
}
println bucks
```



src/eg02/04.groovy

Should be Inject:

```
List<Person> people = Person.findAll();

def  frequencies = new HashMap<String, List<Person>>();
for (Person p : people) {
    if (frequencies.containsKey(p.getLastName())) {
        frequencies.get(p.getLastName()).add(p)
    } else {
        frequencies.put(p.getLastName(), [p])
    }
}
println frequencies
```

src/eg02/05.groovy

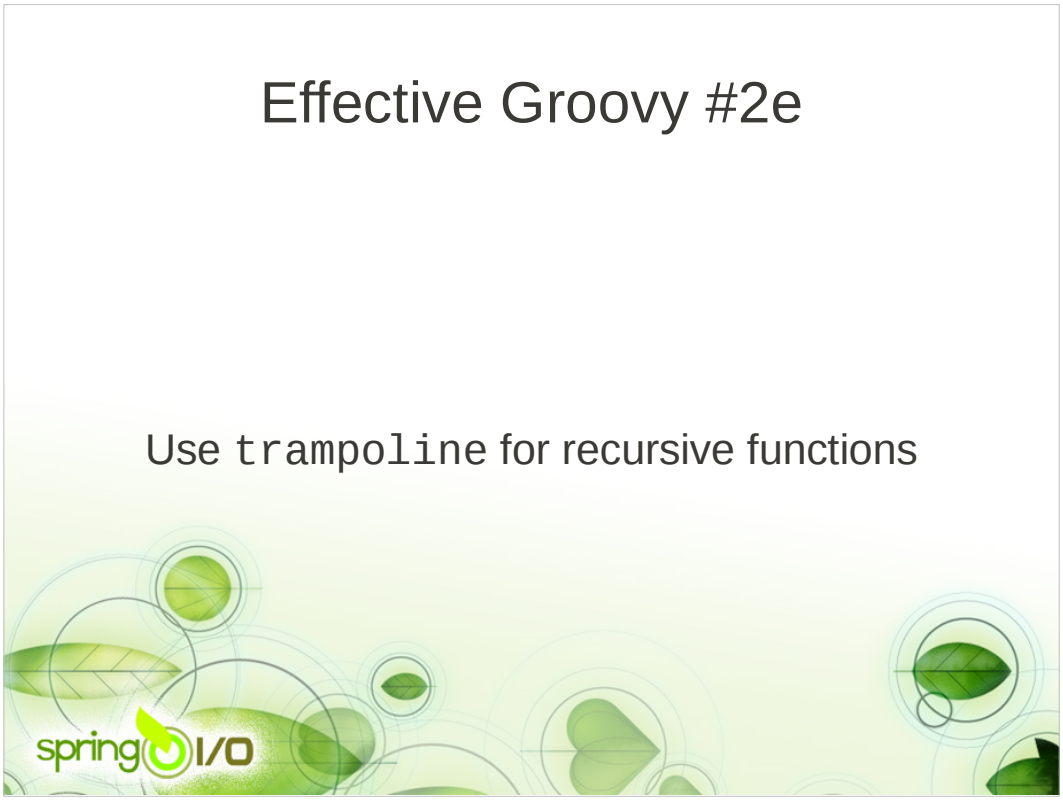
Should be Unique:

```
List<Person> people = Person.findAll();

List<Person> familyReps = new ArrayList<Person>();
List<String> seenFamilies = new ArrayList<String>();
for (Person p : people) {
    if (!seenFamilies.contains(p.getLastName())) {
        seenFamilies.add(p.getLastName());
        familyReps.add(p)
    }
}
println familyReps
```

src/eg02/06.groovy

Should be Memoize:

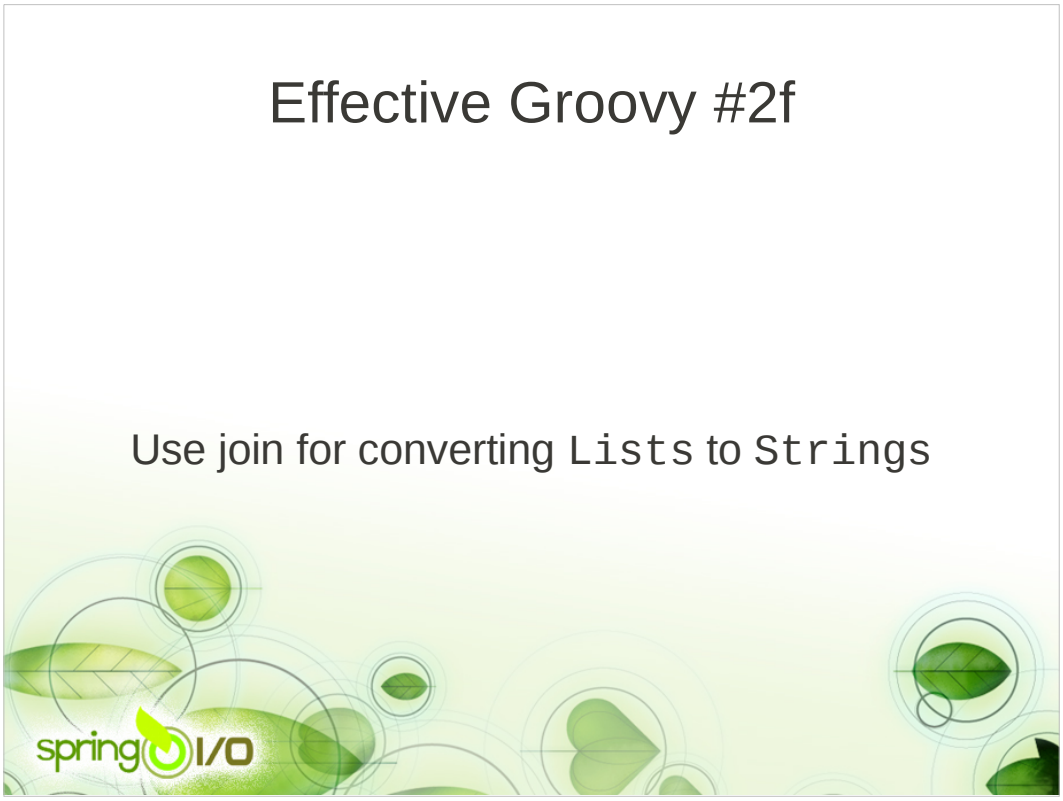
```
def cache = new HashMap<Integer, Integer>()
int fib2(int seed) {
    if (seed == 0) return seed
    if (seed == 1) return seed
    int minus2 = cache.get(seed - 2) ?: fib2(seed - 2)
    int minus1 = cache.get(seed - 1) ?: fib2(seed - 1)
    cache.put(seed-2, minus2)
    cache.put(seed-1, minus1)
    minus2 + minus1
}
```



src/eg02/07.groovy

Should be Trampolined:

```
def factorial2
factorial2 = { BigInteger seed ->
    if (seed == 1) return 1;
    return seed * factorial2(seed -1);
}

println factorial2(200)
```



Should be join()

```
List<Person> people = Person.findAll();

String msg = "";
for (Person p : people) {
    msg = msg + p + "\n";
}
System.out.println(msg);
```

src/eg02/09.groovy – Should be any:

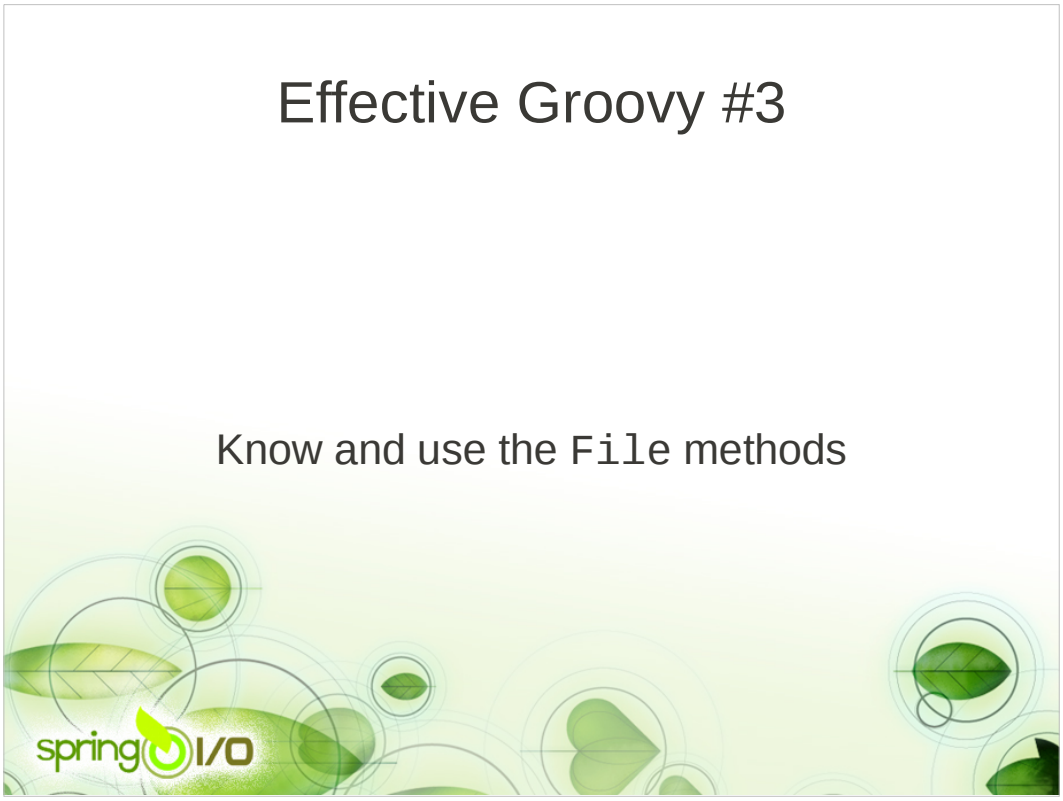
```
List<Person> people = Person.findAll();
boolean found = false;
for (Person p : people) {
    if ("Joe".equals(p.getFirstName())) {
        found = true;
        break;
    };
}
```

src/eg02/10.groovy – Should be all

```
List<Person> people = Person.findAll();

boolean mismatch = true;
for (Person p : people) {
    if (p.getFirstName() == null) {
        mismatch = false;
        break;
    };
}
```

Should be left alone:

```
List<Person> people = Person.findAll();

for (Person p : people) {
    System.out.println(p.firstName);
}
```



Understand the performance implications of your code

01 – should be File.getText() / setText(String)

```
BufferedReader input = new BufferedReader(new
    FileReader("./01.groovy"));
String str;
while ((str = input.readLine()) != null) {
    System.out.println(str);
}
try {input.close();} catch (IOException ex) {}
```

02 – File.readLine – Most File.getText should be readLine with a Closure

03 – File.append – Most File.setText should be File.append

Know the performance implications of I/O!

01 - not thread safe – Show an unsafe class
02 - method synchronization – Add method synchronization. Explain how method synchronization is a bad choice.
03 - internal synchronization – Show synchronization on an internal lock, and how hard it is in Java.
04 - @Synchronized – Show how simple it is. Mention that it exists in Lombok as well. Mention all the CodeNarc and IDEA concurrency rules that help you write properly threaded code.

05 - ReentrantReadWriteLock-Show read/write locks

06 - with Block

Tip: Should write arm block, not finally blocks.

07 - @WithReadLock/@WithWriteLock

Explain some of the CodeNarc rules in this area

Item EG05: Coordination

---

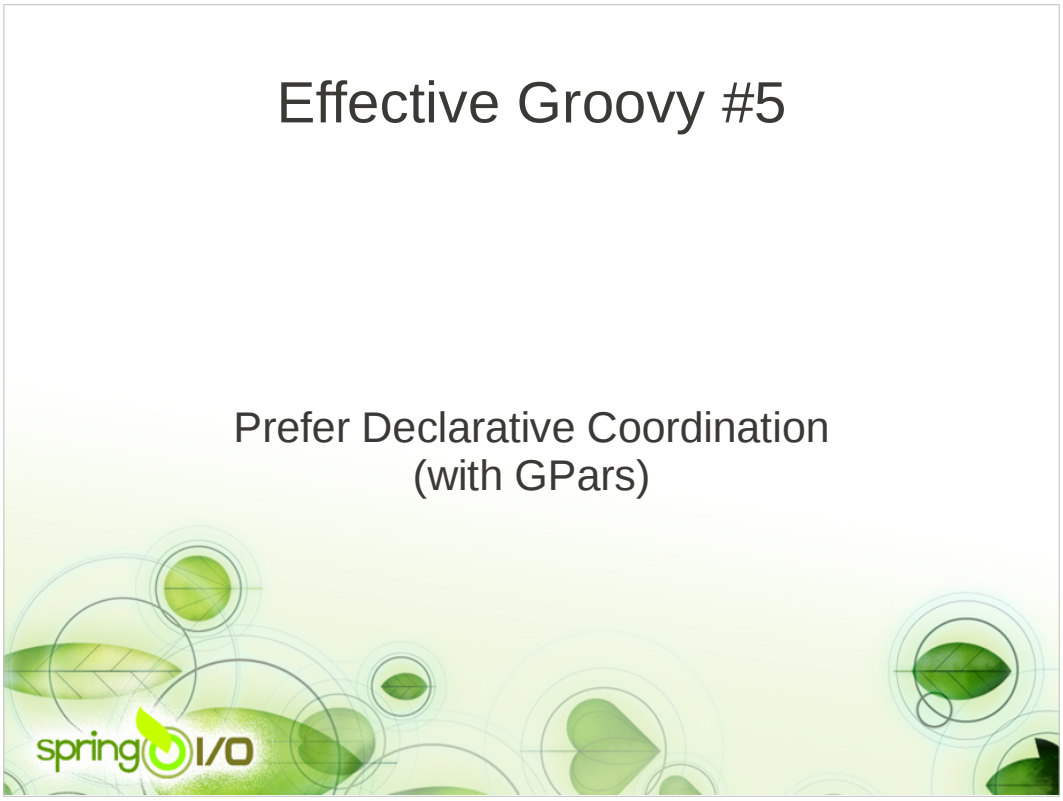
01 - Thread.start and join – Starting threads and waiting on them is easy in Groovy! (but please don't do it)

Item EJ69: Prefer concurrency utilities to wait and notify

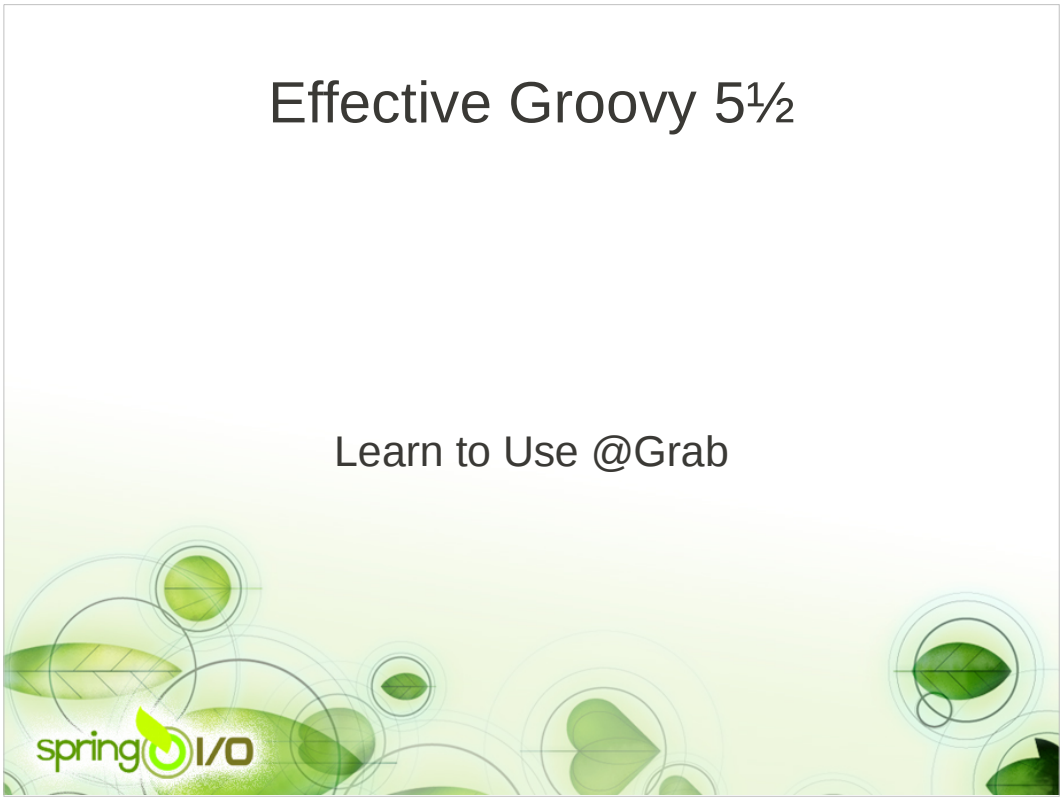
Item EG05: Coordination

---

02 - Prefer executor services - Item EJ68: Prefer Executors to Threads



Item EG05: Coordination
---
03 - GPars Dataflows - Item EG05a: Prefer GPars
04 - Coordination via dataflows



If you are messing around with the classpath then just STOP IT ALREADY [note to self] really scream this one out. Scream you dancing monkey, scream for the audience go on give them what they want.

Example EG06

"Selection among overloaded methods is static, while selection among overridden methods is dynamic. The correct version of an overridden method is chosen at runtime based on the runtime type of the object on which the method is invoked."

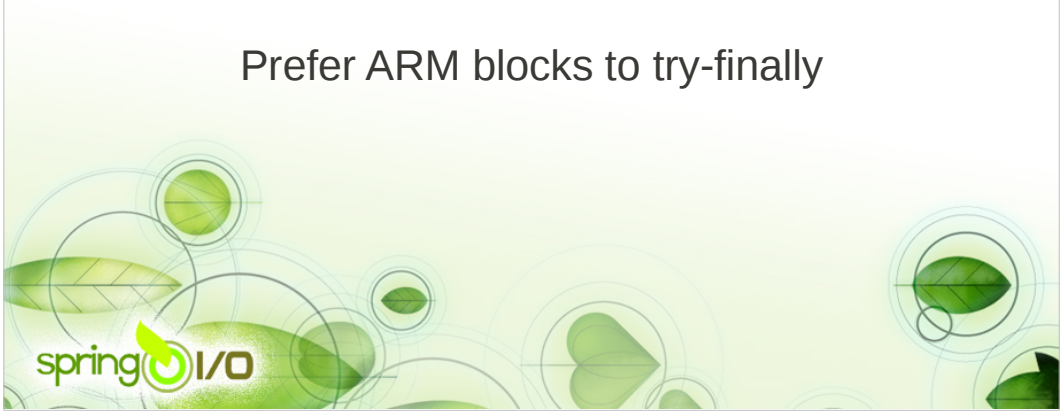
Groovy does this at Runtime

"A safe, conservative policy is never to export two overloadings with the same number of parameters"
Overloaded methods -> Default Parameters
Understand that it creates overloaded methods.
But understand the dispatch happens at runtime.

I put them only in tests

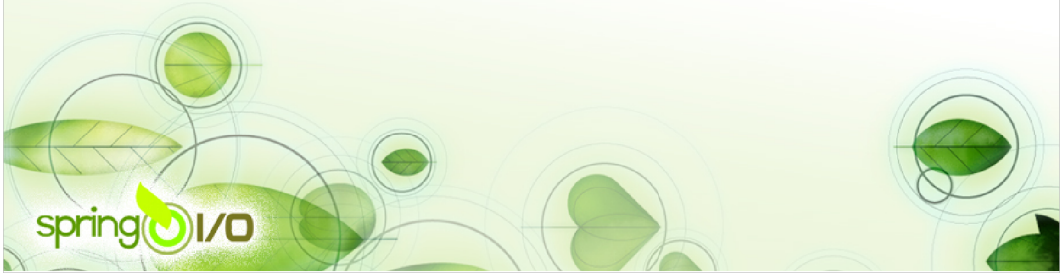
Effective Groovy #8

Prefer ARM blocks to try-finally



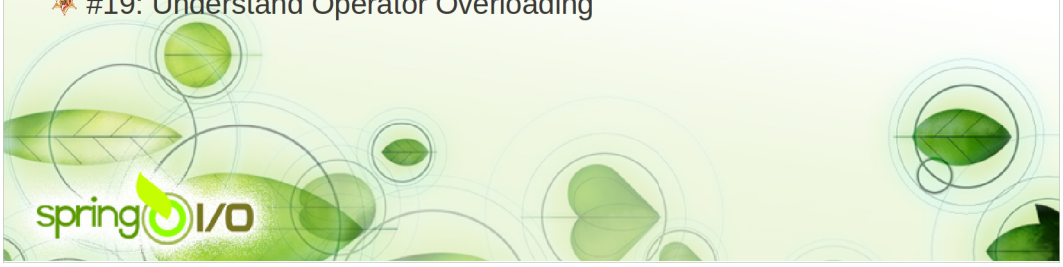
More Effective Java

- #21: Use function objects to represent strategies
- #19: Use interfaces only to define types
- #36: Consistently use @Override
- #43: Return empty Arrays or Collections, not nulls
- #71: Use Lazy Initialization judiciously – See Groovy's @Lazy
- #47: Know & use the libraries – Read the GDK Docs and Release Notes
- #12: Consider implementing Comparable - Important b/c of GroovyTruth
- #63: Include Failure-capture information in detailed messages
- #16: Favor Composition over Inheritence – See Groovy's @Delegate
- #11: Override Clone Judiciously – See @AutoClone, @Canonical

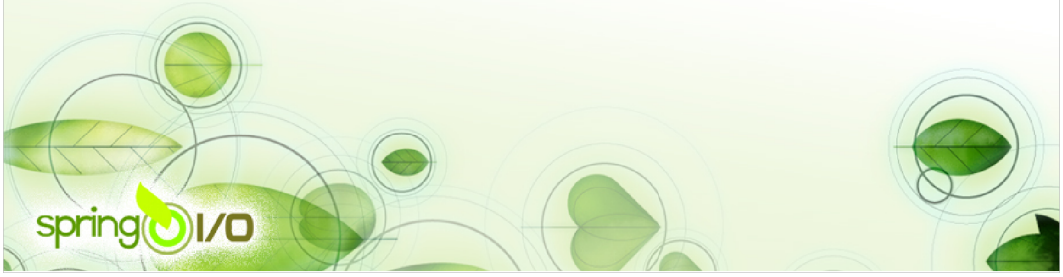


More Effective Groovy

- #9: Learn to Write a Builder
- #10: XMLSlurper/Parser - Do not mix with business logic/layers/etc
- #11: Effective Java #21: Use Function Objects to represent strategies
- #12: Threading: Avoid Busy Wait
- #13: Threading: Avoid Double Checked Locking
- #14: Threading: Avoid Inconsistent Property Locking
- #15: Threading: Avoid Inconsistent Property Synchronization
- #16: Threading: Avoid Synchronizing On Boxed Primitive
- #17: Know and use Elvis operator ?:
- #18: Excessively use the null-safe dereference operator
- #19: Understand Operator Overloading



More Effective Groovy (with static analysis)



canoo

› your provider for business web solutions ›

Groovy, Grails, Griffon, and Java Consulting
info@canoo.com or hamlet.darcy@canoo.com

My Blog - http://hamletdarcy.blogspot.com
Canoo Blog - http://canoo.com/blog
CodeNarc - http://codenarc.sourceforge.net
Twitter for slides: @HamletDRC