Effective Java with Groovy & Kotlin

How Languages Influence Adoption of Good Practices

Naresha K

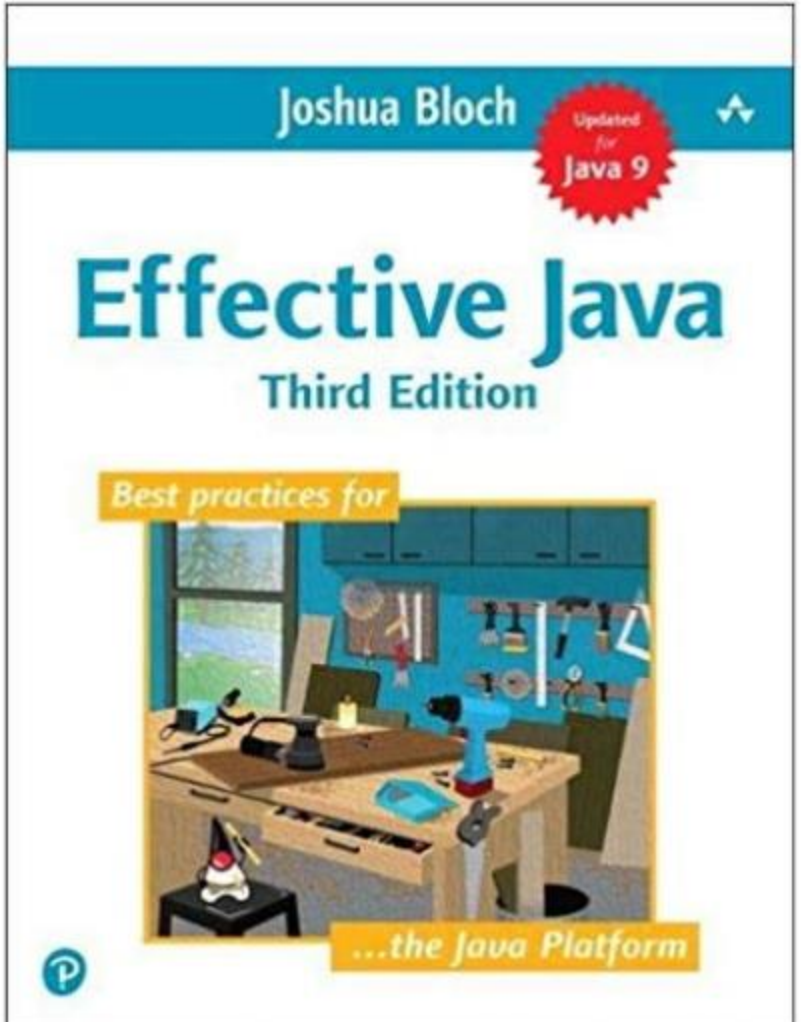
@naresha_k

https://blog.nareshak.com/

About me

Developer, Architect & Coach

Founder & Organiser
Bangalore Groovy User Group


Joshua Bloch
Updated for Java 9
Effective Java
Third Edition
Best practices for
...the Java Platform

http://groovy-lang.org/

initial idea was to make a little dynamic language which compiles directly to Java classes and provides all the nice (alleged) productivity benefits

- James Strachan

Kotlin

https://kotlinlang.org/

Why Kotlin?

Concise

Drastically reduce the amount of boilerplate code.

See example

Safe

Avoid entire classes of errors such as null pointer exceptions.

See example

Interoperable

Leverage existing libraries for the JVM, Android, and the browser.

See example

Tool-friendly

Choose any Java IDE or build from the command line.

See example

https://kotlinlang.org/

YOU ARE HERE

http://commons.wikimedia.org/wiki/File:You-Are-Here-Earth.png

YOU ARE
HERE

we all are

http://commons.wikimedia.org/wiki/File:You-Are-Here-Earth.png

105
106
107
108
109
110
111
112
113
114
116
116
118
118
119
120
120
122
122
124
125
126
127
128
129
131
132
156
155
154
ALL ABOARD
DIAMOND CLUB

Know your guides

Context/ the Problem

What does “Effective Java” say?

The traps

Idiomatic Solution

Lessons Learned

Tool

Wisdom

START HERE

```
class Product {
    String sku
    String description
    BigDecimal price
}
```

```
Product book1 = new Product(
  sku: 'P101',
  description: 'Effective Java',
  price: 40.0)
Product book2 = new Product(
  sku: 'P101',
  description: 'Effective Java',
  price: 40.0)
println book1 == book2
```

false

```
Product book1 = new Product(
  sku: 'P101',
  description: 'Effective Java’,
  price: 40.0)
Product book2 = new Product(
  sku: 'P101',
  description: 'Effective Java’,
  price: 40.0)
def stock = [:]
stock[book1] = 100
println stock[book2]
```

null

protected Object	clone() Creates and returns a copy of this object.
boolean	equals(Object obj) Indicates whether some other object is "e
protected void	finalize() Called by the garbage collector on an obje no more references to the object.
Class<?>	getClass() Returns the runtime class of this Object.
int	hashCode() Returns a hash code value for the object.

#10: Obey the general contract when overriding equals

#11: Always override hashCode when you override equals

EqualsBuilder	Assists in implementing `Object.equals(Object)` methods.
HashCodeBuilder	Assists in implementing `Object.hashCode()` methods.

```
public boolean equals(Object obj) {
  if (obj == null) { return false; }
  if (obj == this) { return true; }
  if (obj.getClass() != getClass()) {
    return false;
  }
  MyClass rhs = (MyClass) obj;
  return new EqualsBuilder()
                .appendSuper(super.equals(obj))
                .append(field1, rhs.field1)
                .append(field2, rhs.field2)
                .append(field3, rhs.field3)
                .isEquals();
 }
```

```
class Product {
    String sku
    String description
    BigDecimal price
}
```

```
class Product {
    String sku
    String description
    BigDecimal price
    LocalDate dateOfManufacture
}
```

```
class Product {
    String sku
    String description
    BigDecimal price
    LocalDate dateOfManufacture
    LocalDate dateOfExpiry
}
```

```
@EqualsAndHashCode
class Product {
    String sku
    String description
    BigDecimal price
}
```

```
Product book1 = new Product(
  sku: 'P101',
  description: 'Effective Java’,
  price: 40.0)
Product book2 = new Product(
  sku: 'P101',
  description: 'Effective Java’,
  price: 40.0)
println book1 == book2
```

true

```
Product book1 = new Product(
  sku: 'P101',
  description: 'Effective Java’,
  price: 40.0)
Product book2 = new Product(
  sku: 'P101',
  description: 'Effective Java’,
  price: 40.0)
def stock = [:]
stock[book1] = 100
println stock[book2]
```

100

```
@EqualsAndHashCode(includes = 'id')
class Product {
    Long id
    String sku
    String description
    BigDecimal price
}
```

```
@EqualsAndHashCode(includes = 'id')
class Product {
    Long id
    String sku
    String description
    BigDecimal price
}
```

```
@EqualsAndHashCode(includes = 'sku')
class Product {
    Long id
    String sku
    String description
    BigDecimal price
}
```

```
class Product(val sku: String, val description: String,
   val price: BigDecimal)

fun main() {
    val book1 = Product("P101", "Effective Java", BigDecimal("35.08")
    val book2 = Product("P101", "Effective Java", BigDecimal("35.08")
    println(book1 == book2)

    val stock = mapOf(book1 to 100)
    println(stock[book2])
}
```

false

null

```
data class Product(val sku: String, val description: String, val price
BigDecimal)

fun main() {
    val book1 = Product("P101", "Effective Java", BigDecimal("35.08")
    val book2 = Product("P101", "Effective Java", BigDecimal("35.08")
    println(book1 == book2)

    val stock = mapOf(book1 to 100)
    println(stock[book2])
}
```

true

100

```
data class Product(val sku: String, val description: String,
  val price: BigDecimal, var id: Long?)

fun main() {
    val book1 = Product("P101", "Effective Java", BigDecimal("35.08"), nu
    val book2 = Product("P101", "Effective Java", BigDecimal("35.08"), 1L
    println(book1 == book2)

    val stock = mapOf(book1 to 100)
    println(stock[book2])
}
```

false

null

```kotlin
data class Product(val sku: String, val description: String, val price
BigDecimal) {
    var id: Long? = null
}

fun main() {
    val book1 = Product("P101", "Effective Java", BigDecimal("35.08"))
    val book2 = Product("P101", "Effective Java", BigDecimal("35.08"))
    book2.id = 1L
    println(book1 == book2)

    val stock = mapOf(book1 to 100)
    println(stock[book2])
}
```

true

100

AST Transformation

Language / Compiler

Single point of representation of any knowledge

#58: Prefer for-each loops to traditional for loops

```
def numbers = [10, 20, 30, 40]
def sum = 0
for(int number in numbers){
    sum += number
}
println "Sum: " + sum
```

```
numbers.each { number ->
    println number
}
```

```
10
20
30
40
```

```
println numbers.collect { number ->
    number * 2
}
```

```
[20, 40, 60, 80]
```

```
println numbers.inject(0,
        { result, number -> result + number }
)
```

```
100
```

```
numbers.forEach {
    println(it)
}
```

10
20
30
40

```
println(numbers.map {
    it * 2
})
```

[20, 40, 60, 80]

```
println(numbers.fold(0)
   { acc, number -> acc + number })
```

100

Closures/ Higher Order Functions

Favor Internal iterators to external iterators

Minimise the moving parts

```java
float price = 0.1f;
float total = 0;
for(int i=0; i<10; i++){
    total += price;
}
System.out.println("Total: " + total);
```

Total: 1.0000001

```java
double price = 0.1;
double total = 0;
for(int i=0; i<10; i++){
    total += price;
}
System.out.println("Total: " + total);
```

Total: 0.9999999999999999

#60: Avoid float and double if exact answers are required

```java
BigDecimal price = new BigDecimal(0.1);
BigDecimal total = new BigDecimal(0);
for(int i=0; i<10; i++){
    total = total.add(price);
}
System.out.println("Total: " + total);
```

Total: 1.000000000000000055511151231257827021181583404541015625 0

```
BigDecimal price = new BigDecimal(0.1);
BigDecimal total = new BigDecimal(0);
for(int i=0; i<10; i++){
    total = total.add(price);
}
System.out.println("Total: " + total);
```

Total: 1.000000000000000055511151231257827021181583404541015625 0

```
def price = 0.1
def total = 0
for(int i=0; i<10; i++){
    total += price
}
println "Total: " + total
```

Total: 1.0

```
def price = 0.1
def total = 0
for(int i=0; i<10; i++){
    total += price
}
println "Total: " + total
println price.class
println total.class
```

Total: 1.0
class java.math.BigDecimal
class java.math.BigDecimal

Select appropriate defaults

Principle of Least Astonishment

Million Dollar Effort

Million Dollar Effort

null check

```
List<Speaker> getSpeakers(String conference) {
    return null;
}
List<Speaker> j2DaysSpeakers = getSpeakers("Java2Days-2019"
if (j2DaysSpeakers != null) {
    //...
}
```

#54: Return empty arrays or collections, not nulls

```
println getSpeakers('Java2Days-2019')
        .collect { it.firstName }
println getSpeakers('Java2Days-2019')
        .findAll { it.firstName.length() > 5 }
```

```
println getSpeakers('Java2Days-2019')
        .collect { it.firstName } // []
println getSpeakers('Java2Days-2019')
        .findAll { it.firstName.length() > 5 } // []
```

org.codehaus.groovy.runtime

Class NullObject

```
public Iterator iterator()
```

iterator() method to be able to iterate on null. Note: this part is from
Invoker

Returns:

an iterator for an empty list

```kotlin
fun getSpeakers(conference: String): List<Speaker> {
    return null
}
```

```
fun getSpeakers(conference: String): List<Speaker> {
    return null
}
```

Null can not be a value of a non-null type List<Speaker>

```
fun getSpeakers(conference: String): List<Speaker> {
    return null
}
```

Null can not be a value of a non-null type List<Speaker>

```
fun getSpeakers(conference: String): List<Speaker>? {
    return null
}
```

NullObject Pattern

Type system

No boilerplate code - $$$

Life is too short for null checks!

Design Patterns
Elements of Reusable
Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

#3: Enforce the singleton property with a private constructor or an enum type

```
class Manager {
    private static final Manager manager =
            new Manager()

    private Manager() { super() }

    static Manager getInstance() { manager }
}
```

```
def m1 = Manager.getInstance()
def m2 = Manager.getInstance()
println m1 == m2
```

true

```
class Manager {
    private static final Manager manager =
            new Manager()

    private Manager() { super() }

    static Manager getInstance() { manager }
}
```

```
def m1 = Manager.getInstance()
def m2 = Manager.getInstance()
println m1 == m2
```

true

```
def m3 = new Manager()
println m3
```

Manager@b968a76

```
@Singleton
class Manager {}
```

```
def m1 = Manager.getInstance()
def m2 = Manager.getInstance()
println m1 == m2

def m3 = new Manager()
println m3
```

true

Caught: java.lang.RuntimeException: Can't instantiate singleton Manager. Use Manager.instance

```kotlin
object Manager {
    init {
        println("Initialising Manager")
    }
}
```

```java
public final class Manager {
   public static final Manager INSTANCE;

   private Manager() {
   }

   static {
      Manager var0 = new Manager();
      INSTANCE = var0;
      String var1 = "Initialising Manager";
      boolean var2 = false;
      System.out.println(var1);
   }
}
```

```
private static Manager manager

static Manager getInstance() {
    if(!manager){ manager = new Manager() }
    manager
}
```

```
private static Manager manager

static synchronized Manager getInstance() {
    if(!manager){ manager = new Manager() }
    manager
}
```

```
private static volatile Manager manager

static synchronized Manager getInstance() {
    if(!manager){ manager = new Manager() }
    manager
}
```

```
@Singleton(lazy=true)
class Manager {}
```

```
public static Manager getInstance() {
    if (instance != null) {
        return instance
    } else {
        synchronized (Manager) {
            if (instance != null) {
                return instance
            } else {
                return instance = new Manager()
            }
        }
    }
}
```

AST Transformation

Object Declaration

YAGNI

Premature optimisation is the root of all evil

THE PHOENIX CENTER
WEIRD IS A
SIDE EFFECT OF
AWESOME
CITY OF PHOENIX 1202 N. THIRD ST.

#17: Minimize Mutability

Rules to make a class immutable

1. Don’t provide any mutators
2. Ensure that the class can’t be extended
3. Make all fields final
4. Make all fields private
5. Ensure exclusive access to any mutable components

```
class ImmutableClass{
    private final def field1
    private final def field2
    //...
    private final def field10

    public ImmutableClass(f1, f2,… f10){
        //initialization
    }
}
```

```groovy
import groovy.transform.Immutable

@Immutable
class Rectangle {
    int length
    int breadth
}
```

```groovy
def r = new Rectangle(length: 10, breadth: 5)
println r // Rectangle(10, 5)
```

```
public final class Rectangle extends java.lang.Object
             implements groovy.lang.GroovyObject {

    private final int length
    private final int breadth

    public Rectangle(int length, int breadth) {
        //
    }

    public Rectangle(java.util.Map args) {
    }

    public Rectangle() {
        this([:])
    }
}
```

```
new Rectangle(length: 10, breadth:
```

```kotlin
data class Rectangle(val length: Int, val breadth: Int)
```

```java
public final class Rectangle {
   private final int length;
   private final int breadth;

   public final int getLength() {
      return this.length;
   }

   public final int getBreadth() {
      return this.breadth;
   }

   public Rectangle(int length, int breadth) {
      this.length = length;
      this.breadth = breadth;
   }
  // more code
}
```

```kotlin
data class Rectangle(val length: Int, val breadth: Int)

fun main() {
    val rectangle = Rectangle(20, 10)
    val r2 = Rectangle(length = 20, breadth = 10)
}
```

```java
public static final void main() {
   Rectangle rectangle = new Rectangle(20, 10
   Rectangle r2 = new Rectangle(20, 10);
 }
```

AST Transformation

Syntactic Sugar

Readability Matters

#18: Favour composition over inheritance

```
def ph = ['919812312345', '4512341234', '19252199916']
 as PhoneNumbers
println ph.find { it == '19252199916'}
println ph.findAll { it.endsWith('4') }
```

```
def ph = ['919812312345', '4512341234', '19252199916']
 as PhoneNumbers
println ph.find { it == '19252199916'}
println ph.findAll { it.endsWith('4') }
println ph.indianNumbers()
```

```
class PhoneNumbers extends ArrayList {

}
```

```
class PhoneNumbers {
    private @Delegate List phoneNumbers

    PhoneNumbers(numbers) {
        phoneNumbers = numbers
    }

    def indianNumbers() {
        phoneNumbers.findAll { it.startsWith('91') }
    }
}
```

```kotlin
class PhoneNumbers(val list: List<String>) :
       List<String> by list {

    fun indianNumbers(): List<String> {
        return list.filter { it.startsWith("91") }
    }
}
```

```
trait CanSing {
 def sing() {
  println "Singing"
 }
}
```

```
trait CanDance {
 def dance() {
  println "Dancing"
 }
}
```

```
class Person implements CanSing, CanDance {}

Person reema = new Person()
reema.sing()
reema.dance()
```

https://www.slideshare.net/nareshak/designing-with-groovy-traits-gr8conf-india

AST Transformation

Simplify

Take Aways

Some of the ‘Effective Java’ alread
built into the languages

Favour compiler generated cod to IDE generated code

Don’t fall into the trap of copying th
Java implementation

Programming languages can reduc
friction to implement good practice

There could be multiple right solutions. Choose what fits your context

The way we code is influenced no
just by the language we code in,
but also by the languages we
know.

Happy Coding. Thank You