



Universidade Estadual de Maringá (UEM)
Departamento de Informática (DIN)



Frameworks Java

Prof. Dr. Edson A. Oliveira Junior
edson@din.uem.br
www.din.uem.br/~edson



Agenda

- Ferramentas e Tecnologias Java
- Listagem de Frameworks Java
- Spring MVC

Prof. Dr. Edson A. Oliveira Junior
edson@din.uem.br



Universidade Estadual de Maringá (UEM)
Departamento de Informática (DIN)



Motivação para Adoção de Ferramentas e Tecnologias Java

Prof. Dr. Edson A. Oliveira Junior
edson@din.uem.br
www.din.uem.br/~edson



Survey sobre Ferramentas e Tecnologias Java



- 2040 respostas – Mar a Abr de 2016
- Survey por RebelLabs – Simon Maple
- Disponível em
<https://zeroturnaround.com/rebellabs/reports>
- Vamos aos dados...



Survey sobre Ferramentas e Tecnologias Java



Maven

Over **two in three (68%)** devs use Maven as their main build tool



Over **two in three (68%)** devs use Git as their version control



Java 8

Almost **two in three (62%)** devs use Java 8 in production



Jenkins

Three in five (60%) devs use Jenkins for CI



IntelliJ IDEA

Almost **one in two (46%)** devs use IntelliJ, the most popular IDE in the survey.



Over **two in five (43%)** devs use Spring MVC



Tomcat

Over **two in five (42%)** devs use Tomcat server in production



ORACLE[®] D A T A B A S E

Almost **two in five (39%)** devs use Oracle DB in production



Microservices

Over **one in three (34%)** devs have adopted a microservices architecture



Almost **one in three (32%)** devs use Docker in production



Java EE 7

Over **three in ten (31%)** devs use Java EE 7



Almost **three in ten (29%)** devs use Spring Boot





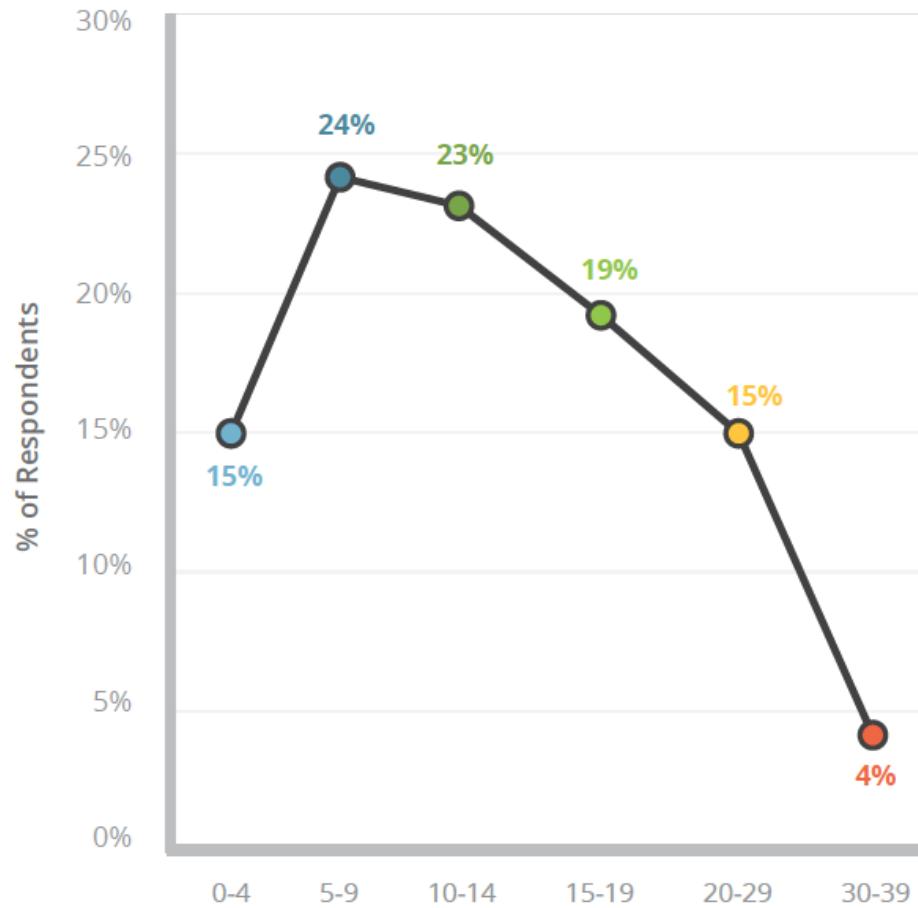
Survey sobre Ferramentas e Tecnologias Java



How many years of experience do you have in software engineering?

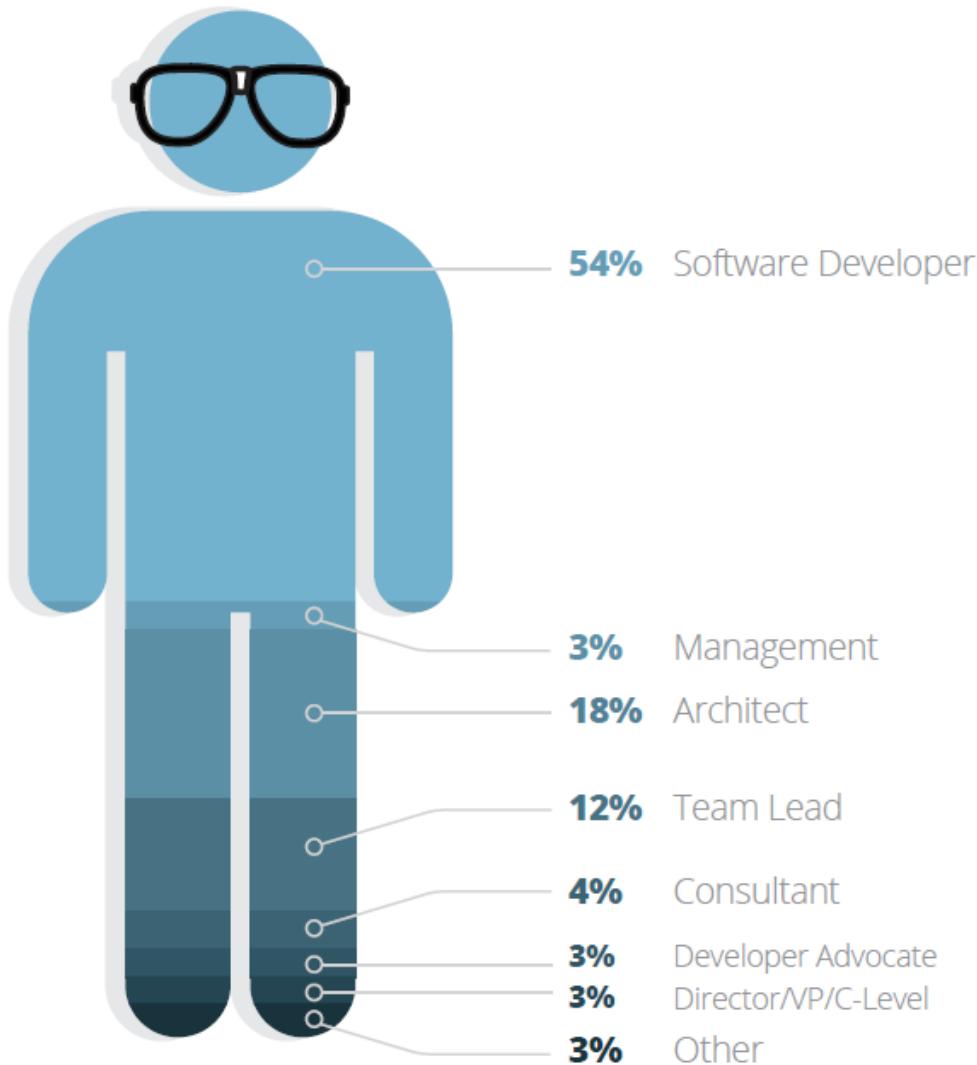
This was a question we've not asked in past surveys. This year, we wanted to understand more about the type of person who responds to the survey as well as their technology preferences. We actually had a pretty good spread of results as you can see from *Figure 1.1*. Around half of the respondents exist in the 5-15 years of experience bracket. The overall median is 10 years of experience and the average is 12.2 years.

Figure 1.1 Years of experience in software engineering





Survey sobre Ferramentas e Tecnologias Java





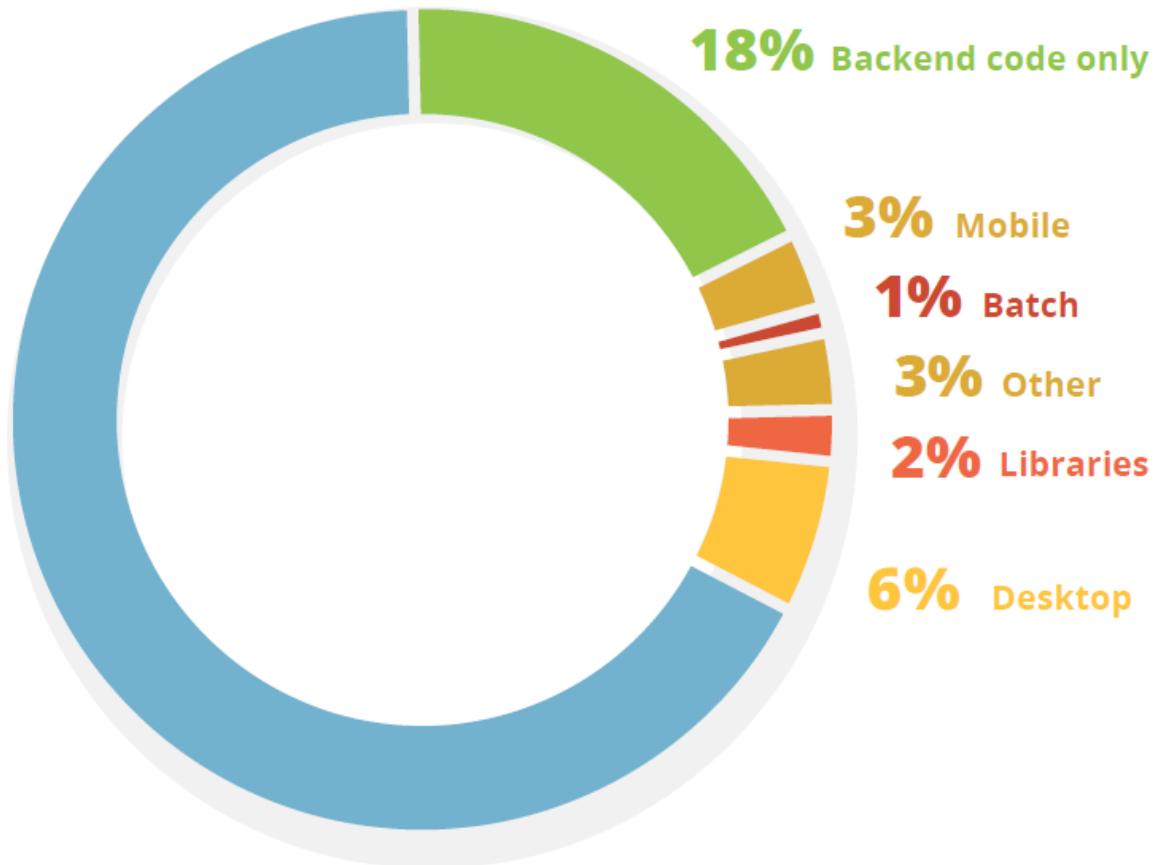
Survey sobre Ferramentas e Tecnologias Java



Which type of application describes
the main project you work on?

We're always keen to get a breakdown of the applications that our respondents work with. As we can see from *Figure 1.6*, two thirds of respondents work on full stack web applications, as you might expect, with a further 18% working on backend code. The remaining votes are split between batch, mobile and libraries, with 6% working on desktop applications. The 'other' option contained middleware developers, tools developers, programming languages and more.

67%
Full stack
web application





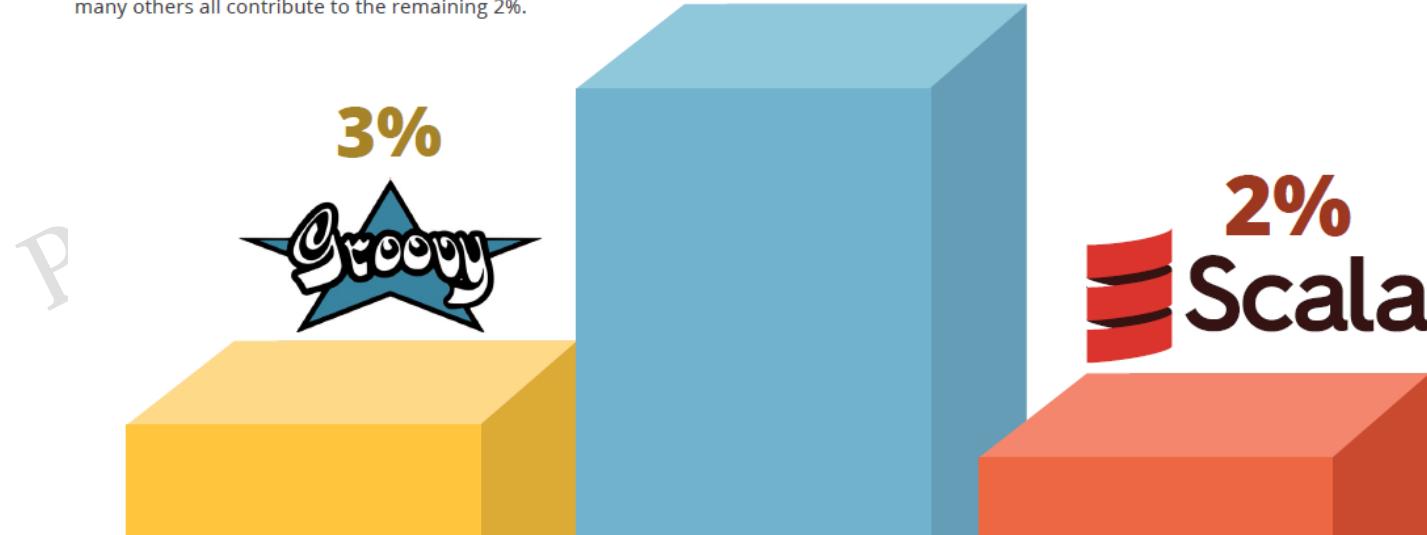
Survey sobre Ferramentas e Tecnologias Java



Which JVM language do you use most often?

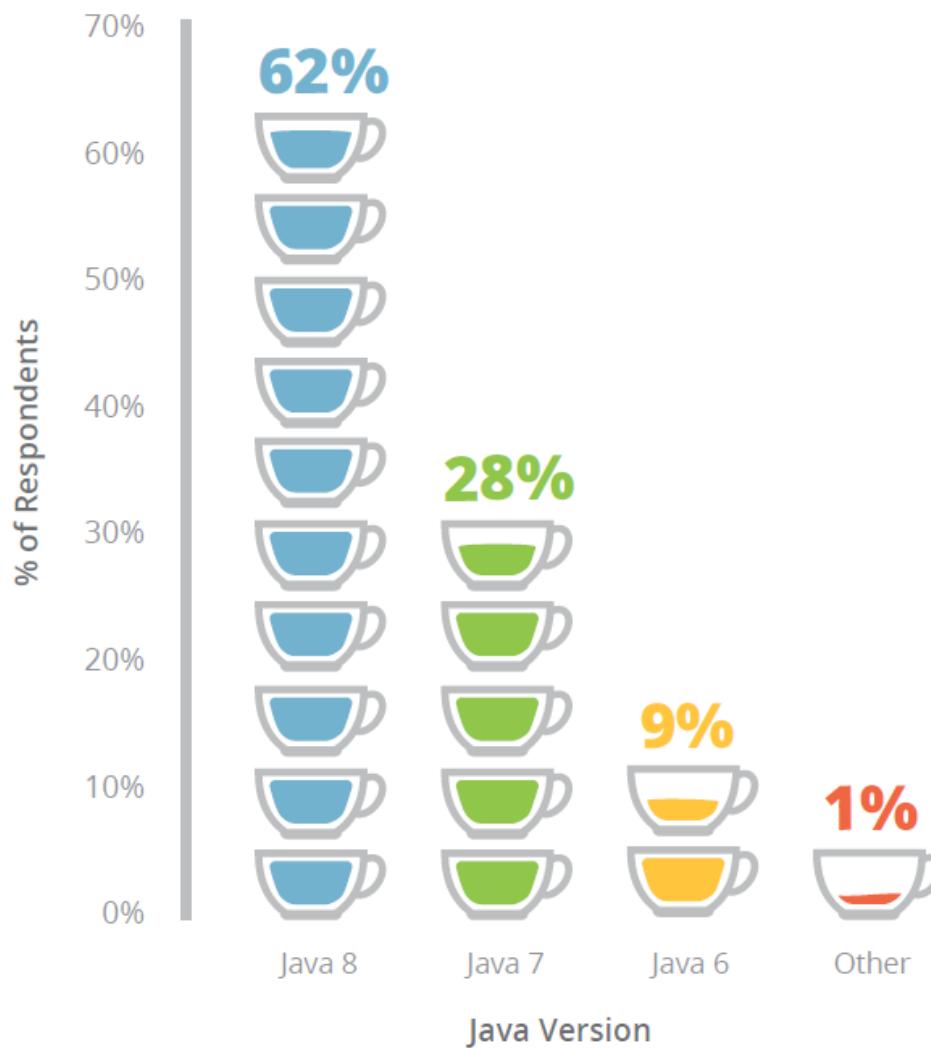
Very often we get a good spread to this question in our survey, but this year shows Java in an extremely dominant lead over other JVM languages.

Notice that we didn't ask which JVM languages people have used, rather which JVM language people use most often. It's clear that while people flirt with other languages in the ecosystem, Java is still monopolizing the JVM with 93% of the total vote. Groovy and Scala, as you'd expect, are next in line — eating the leftovers from Java's table with 3% and 2% respectively. Kotlin, Ceylon, Clojure, JRuby and many others all contribute to the remaining 2%.



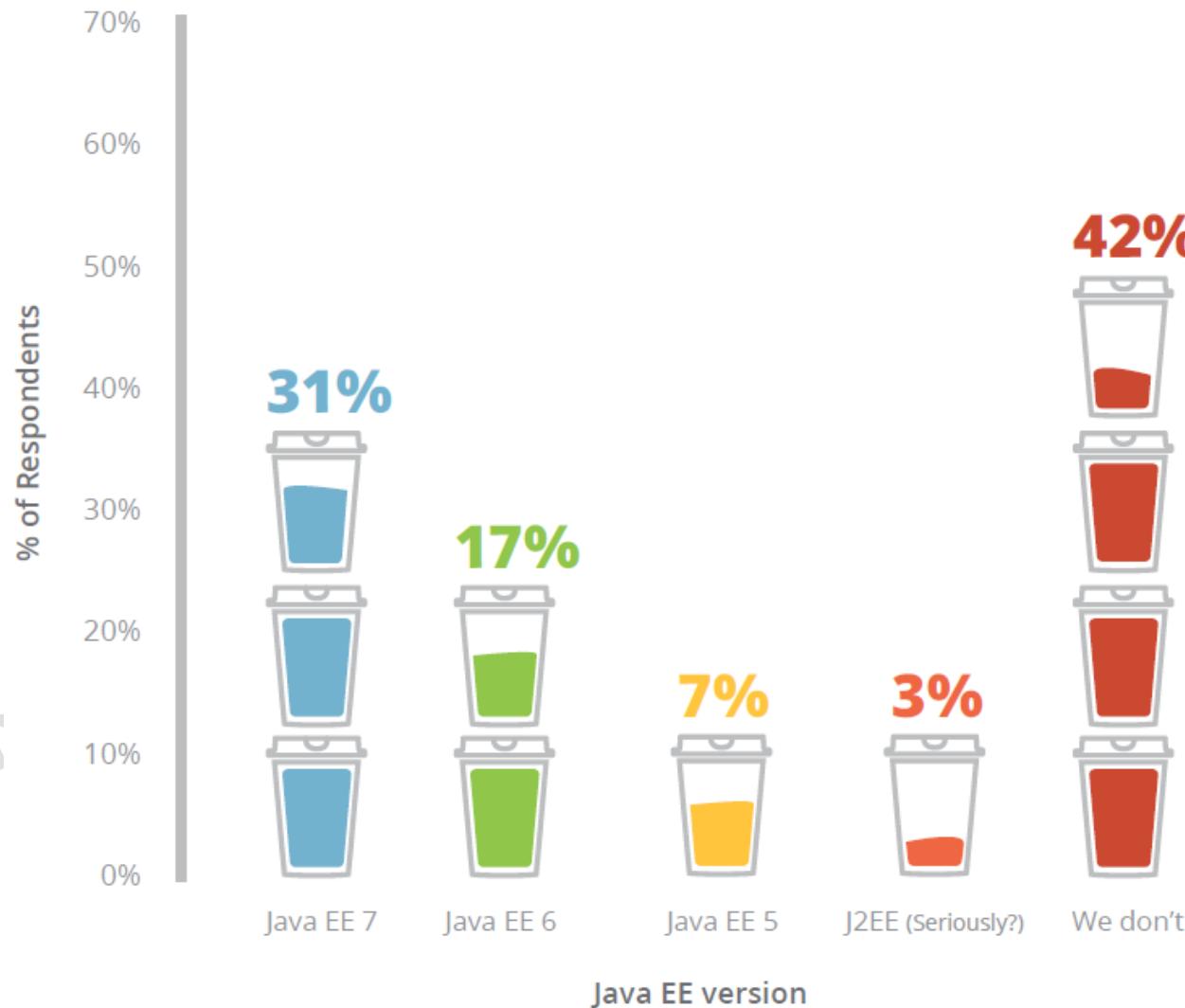


Survey sobre Ferramentas e Tecnologias Java



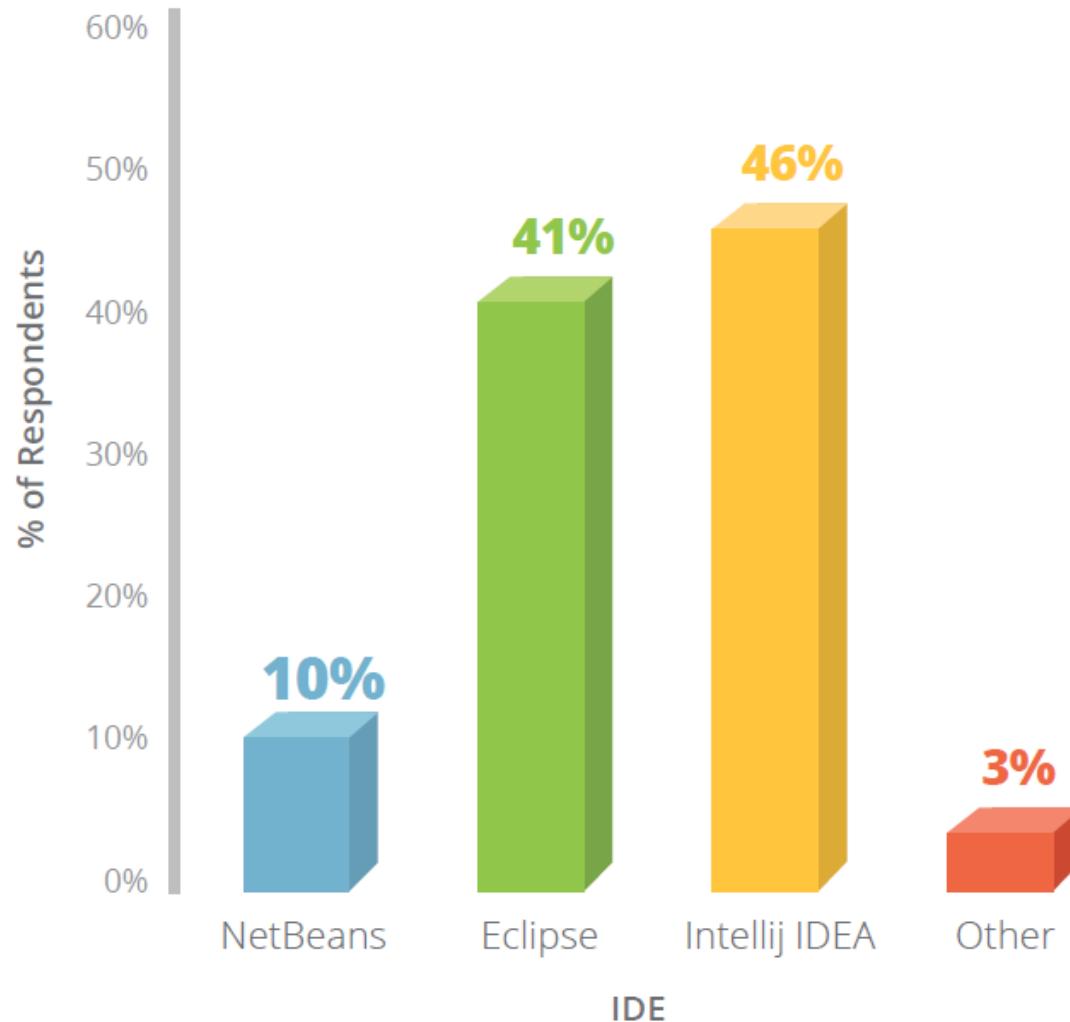


Survey sobre Ferramentas e Tecnologias Java



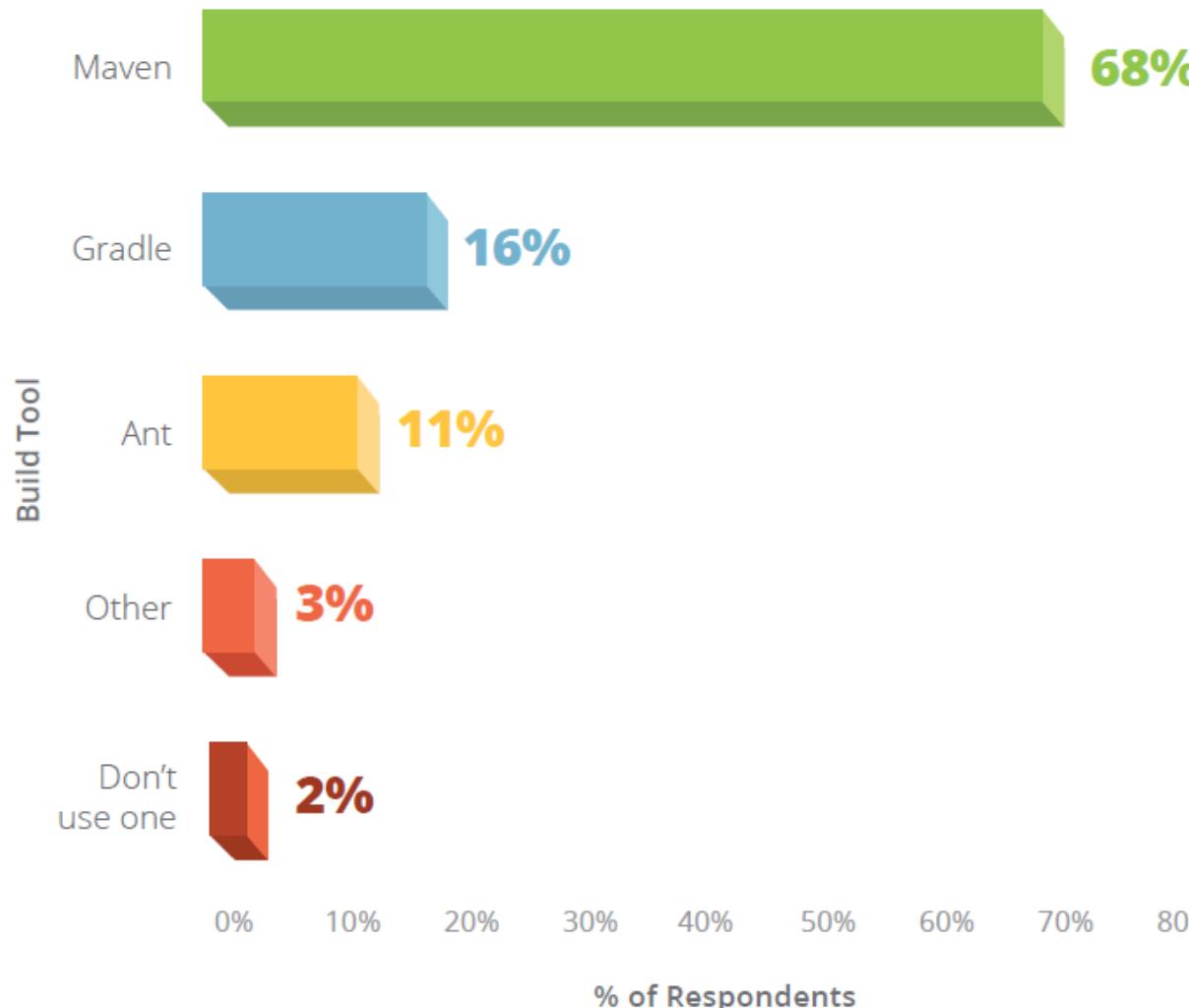


Survey sobre Ferramentas e Tecnologias Java



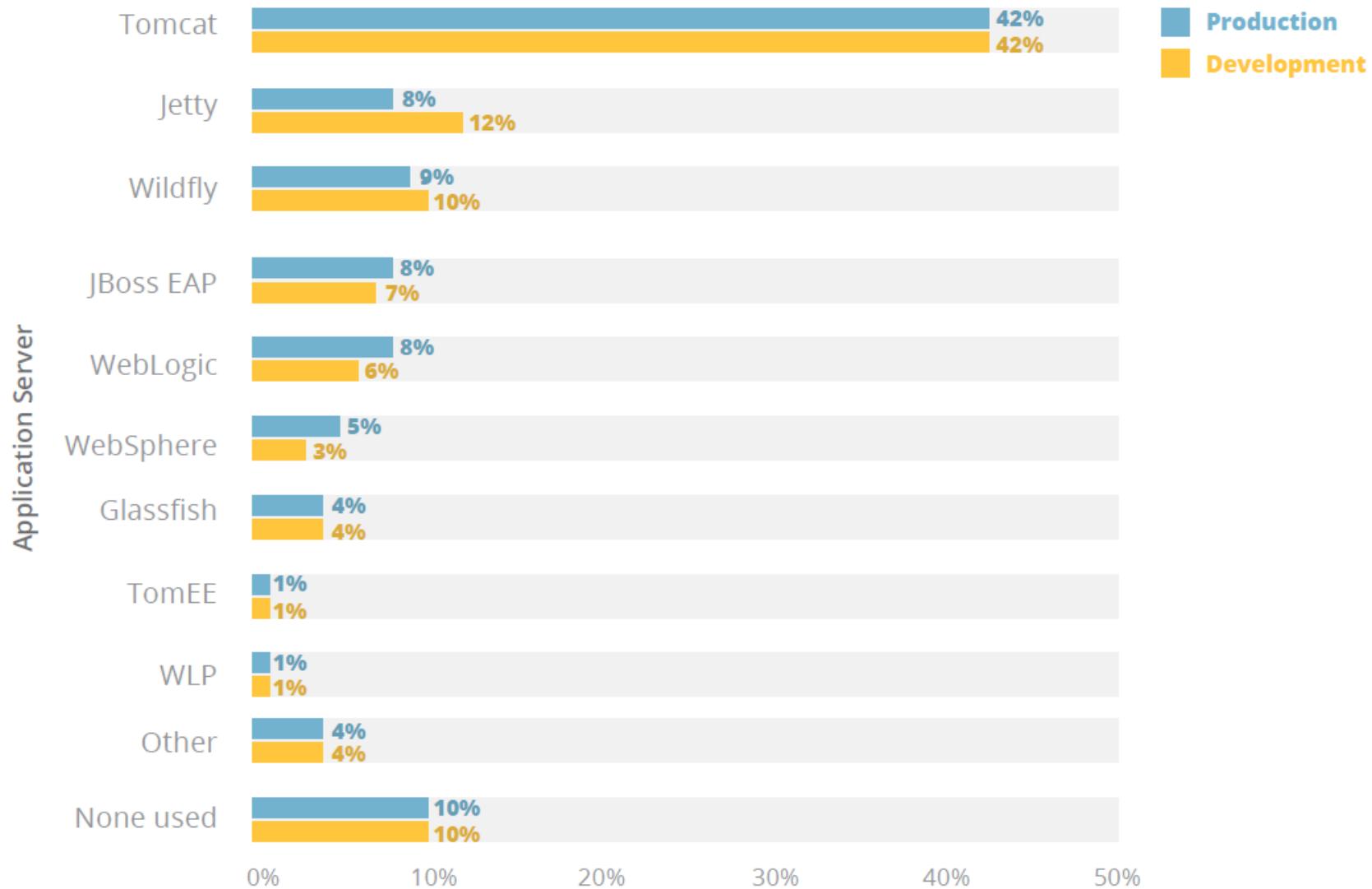


Survey sobre Ferramentas e Tecnologias Java



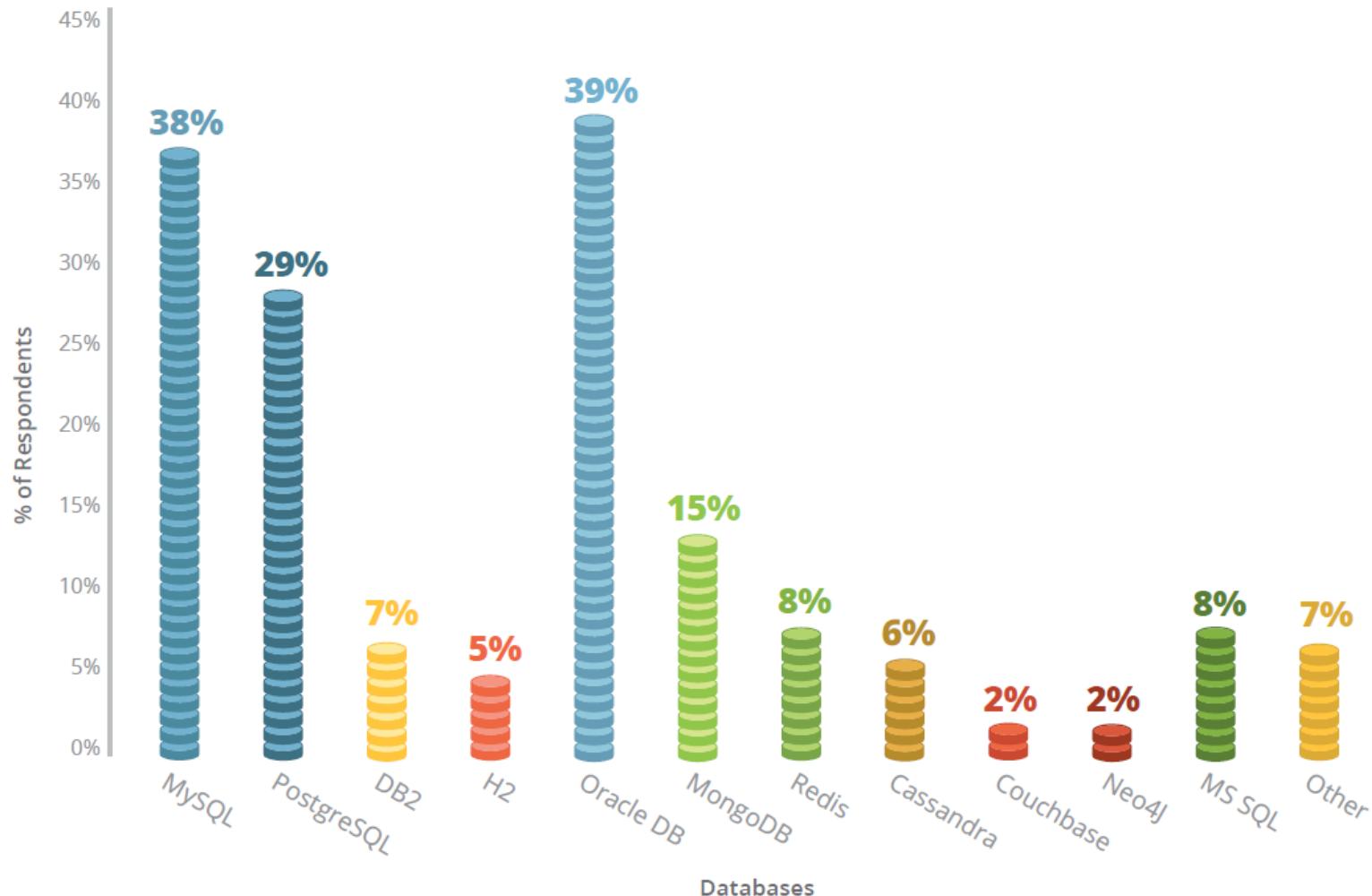


Survey sobre Ferramentas e Tecnologias Java



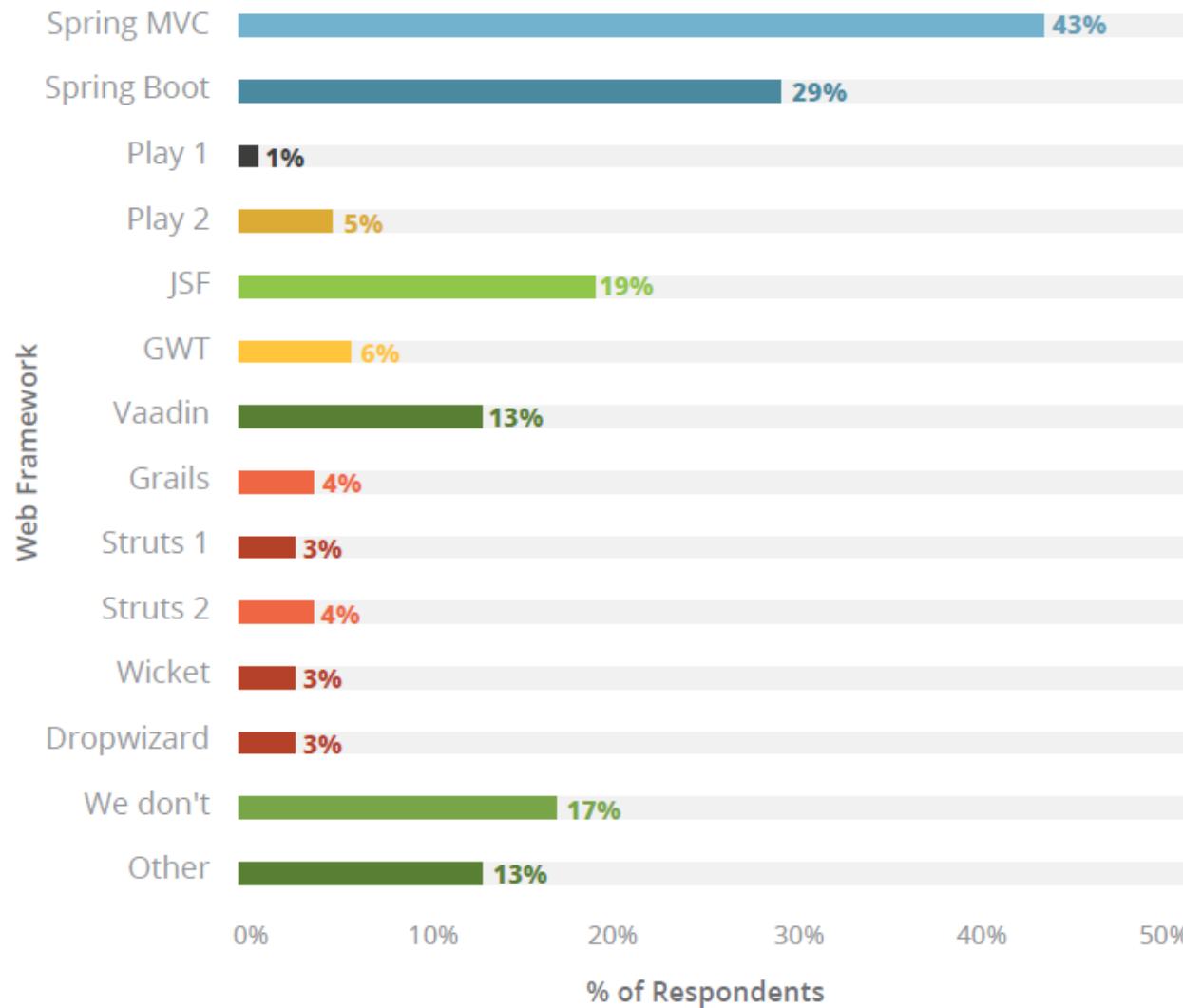


Survey sobre Ferramentas e Tecnologias Java





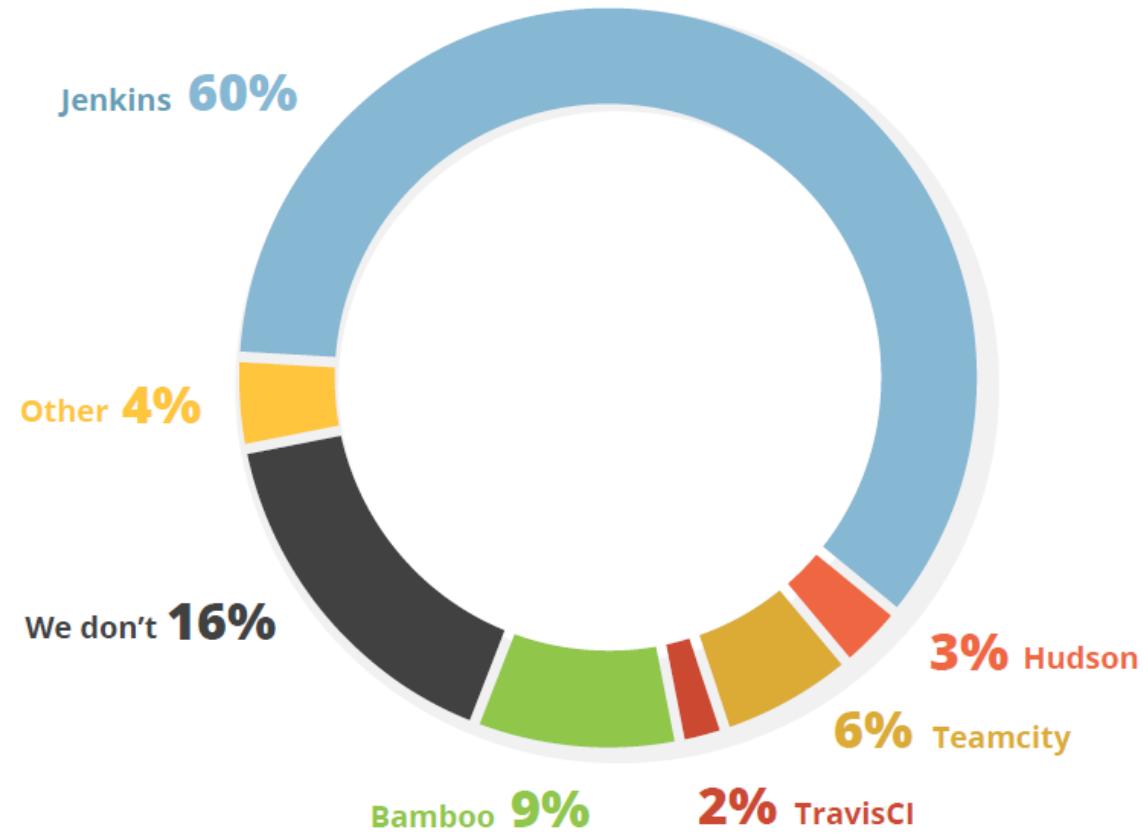
Survey sobre Ferramentas e Tecnologias Java



Which Continuous Integration Server do you use?

As one might expect from a survey that asks which CI servers people use, Jenkins is the runaway winner. The 10-year-old CI server is used by almost two in every three (60%) respondents. The biggest competition by votes is ironically the "We don't do CI" category, which makes Jenkins' dominance even more impressive. Bamboo is the next most popular CI server with 9% of the votes, with TeamCity, Hudson and Travis CI taking up the rest of the share. In the 'other' category, Circle CI is certainly a server worth mentioning that had a good portion of the remaining votes.

Before we go, we should mention Hudson, the CI server from which Jenkins was forked back in 2011, with overwhelming support from the community. Note that Hudson was transferred from Oracle to the Eclipse foundation in 2012. Since Jenkins owns the market in the CI space and with 20 (twenty) times the number of users (using our sample data) compared to Hudson, it's a mystery why Hudson is still being developed or supported today. The game has long been won by Jenkins and perhaps it's time to unplug Hudson altogether.

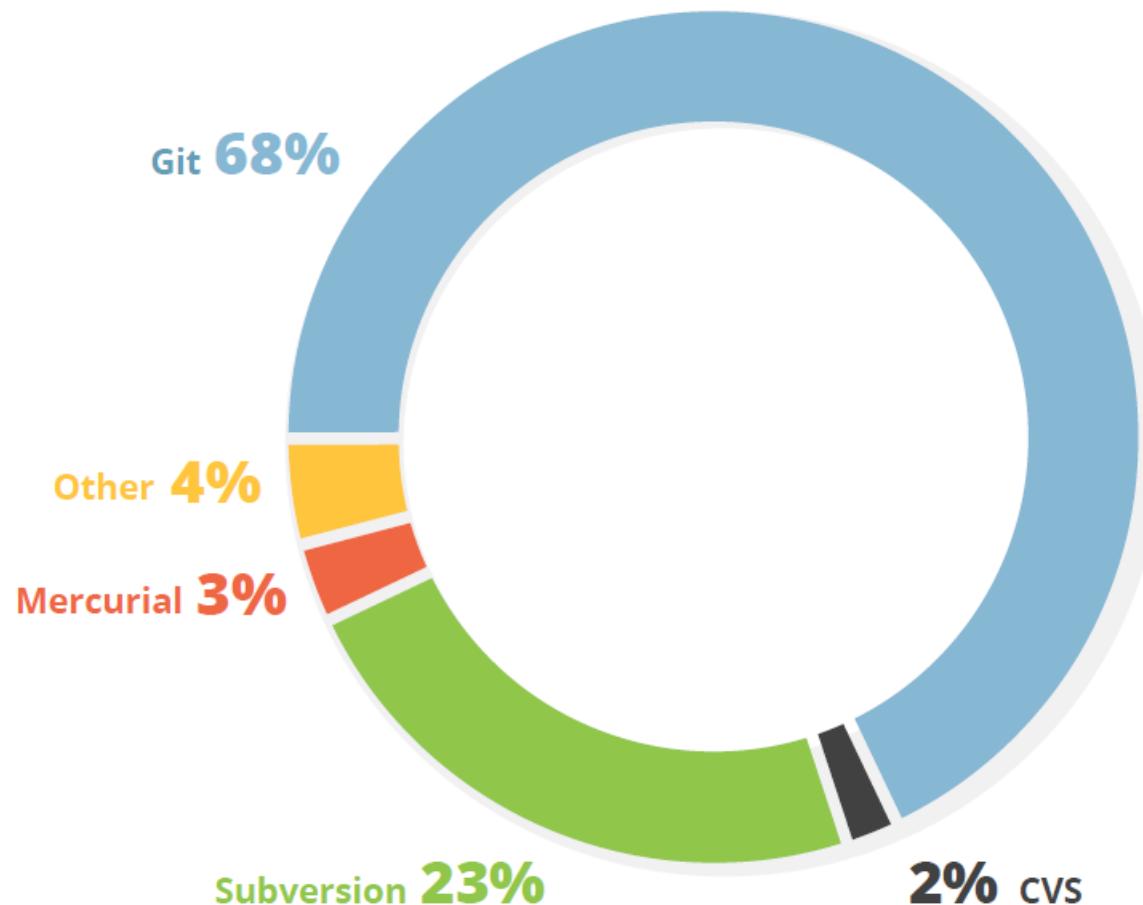




Survey sobre Ferramentas e Tecnologias Java

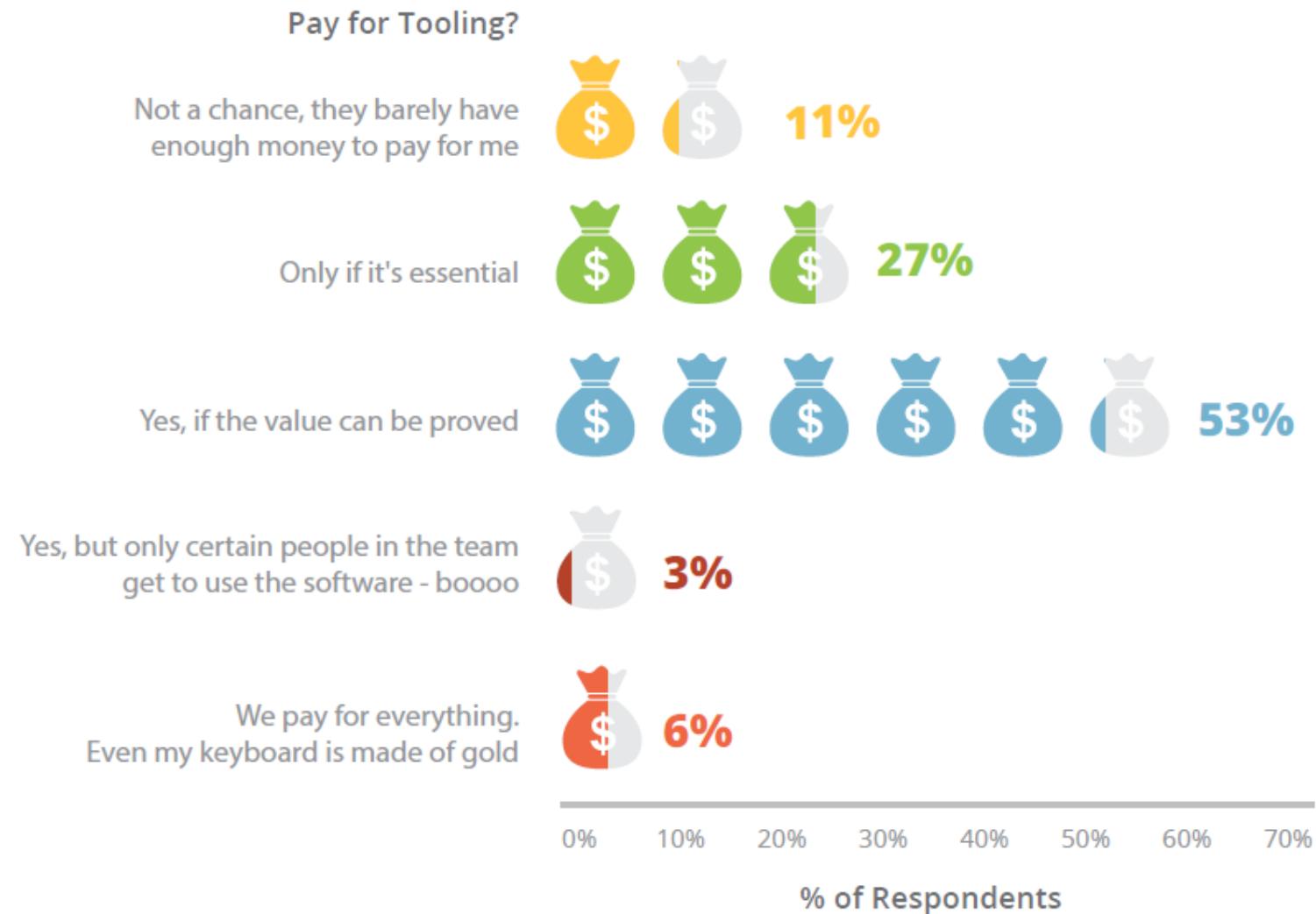


Most Commonly Used VCS





Survey sobre Ferramentas e Tecnologias Java

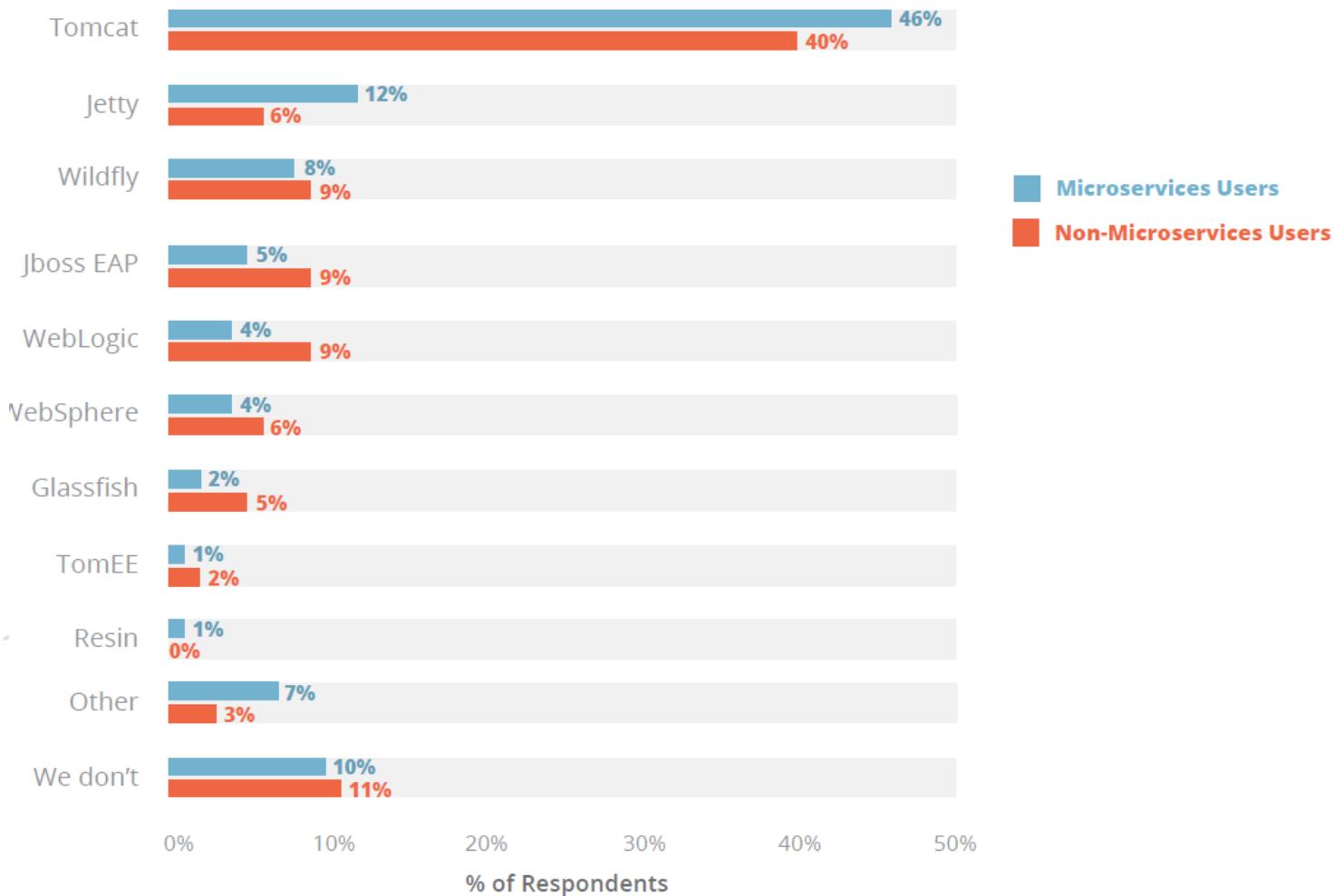




Survey sobre Ferramentas e Tecnologias Java

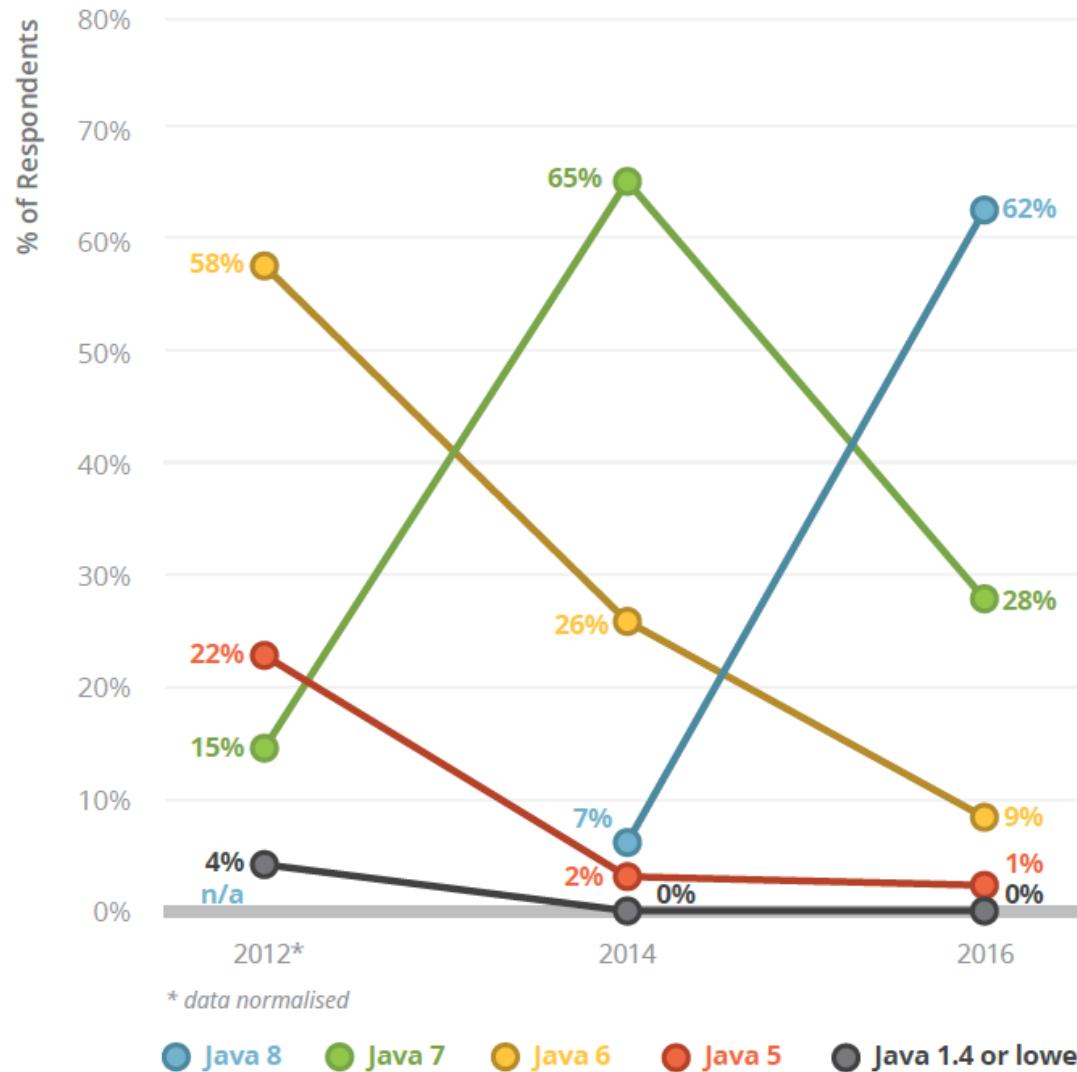


App Server





Survey sobre Ferramentas e Tecnologias Java

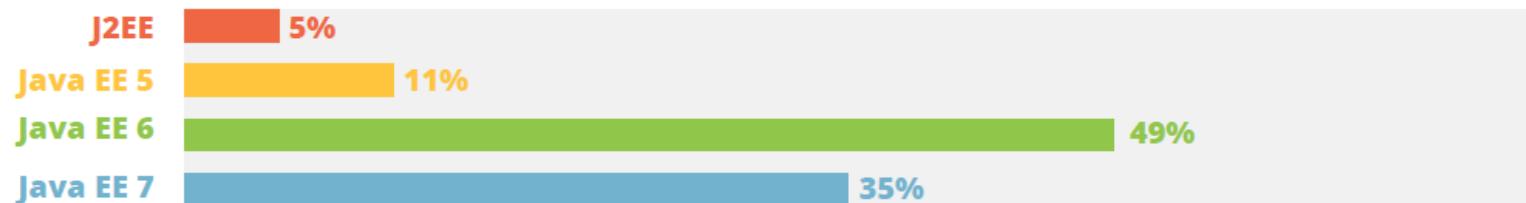




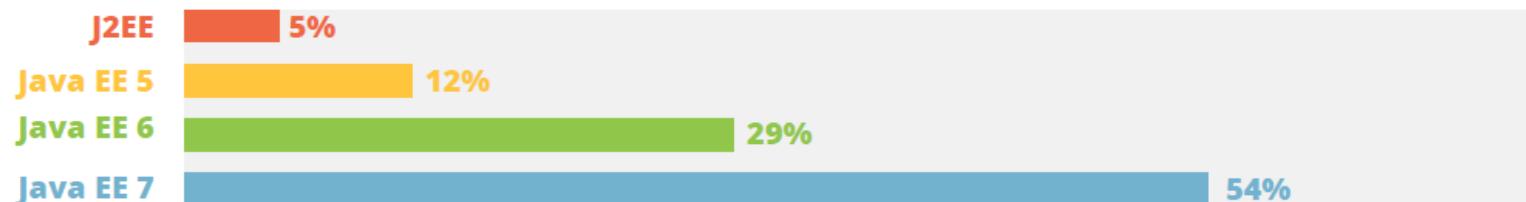
Survey sobre Ferramentas e Tecnologias Java



Java EE Version



2014*



2016*

0%

10%

20%

30%

40%

50%

60%

70%

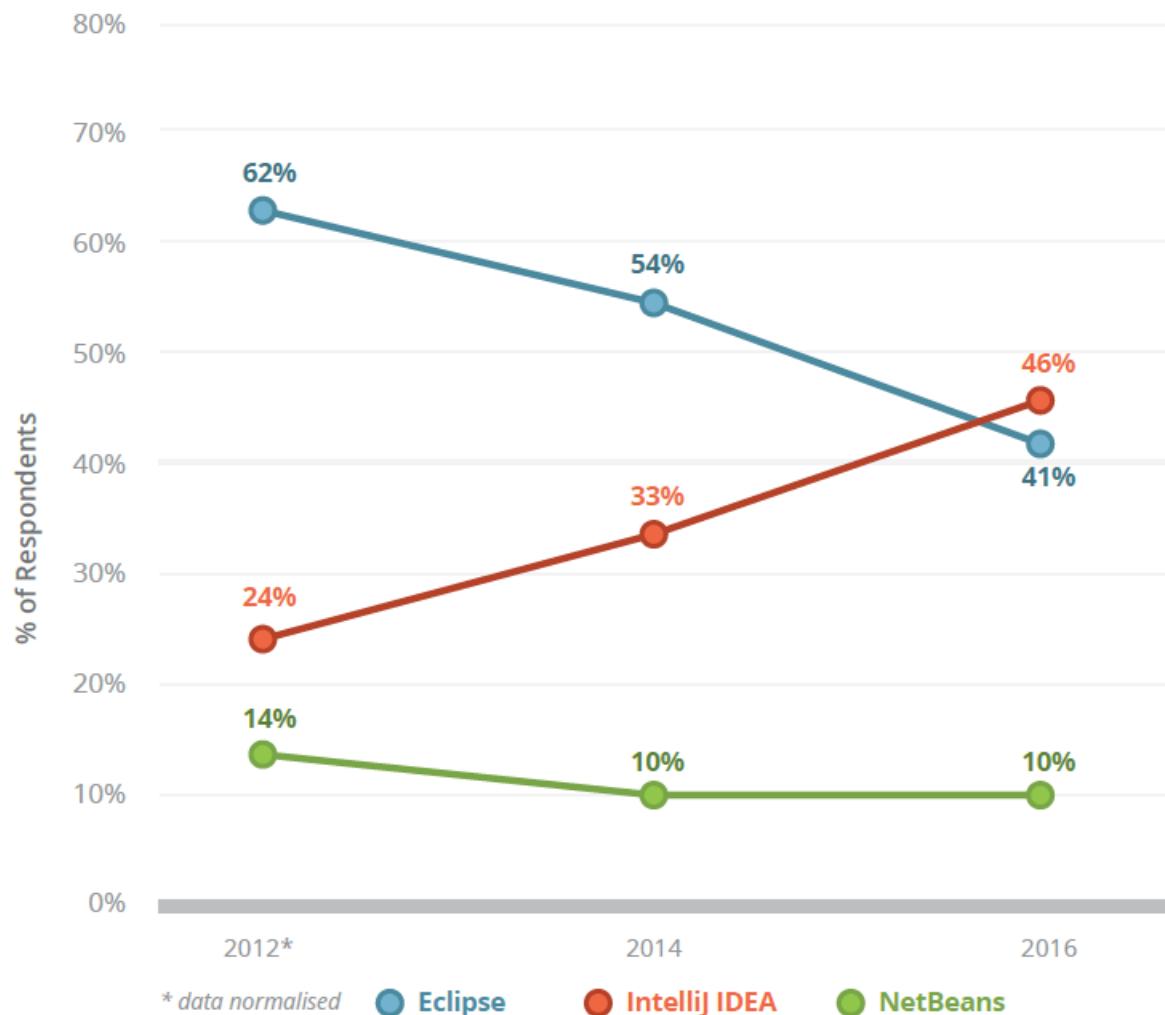
% of Respondents

* data normalised for Java EE users only

YU

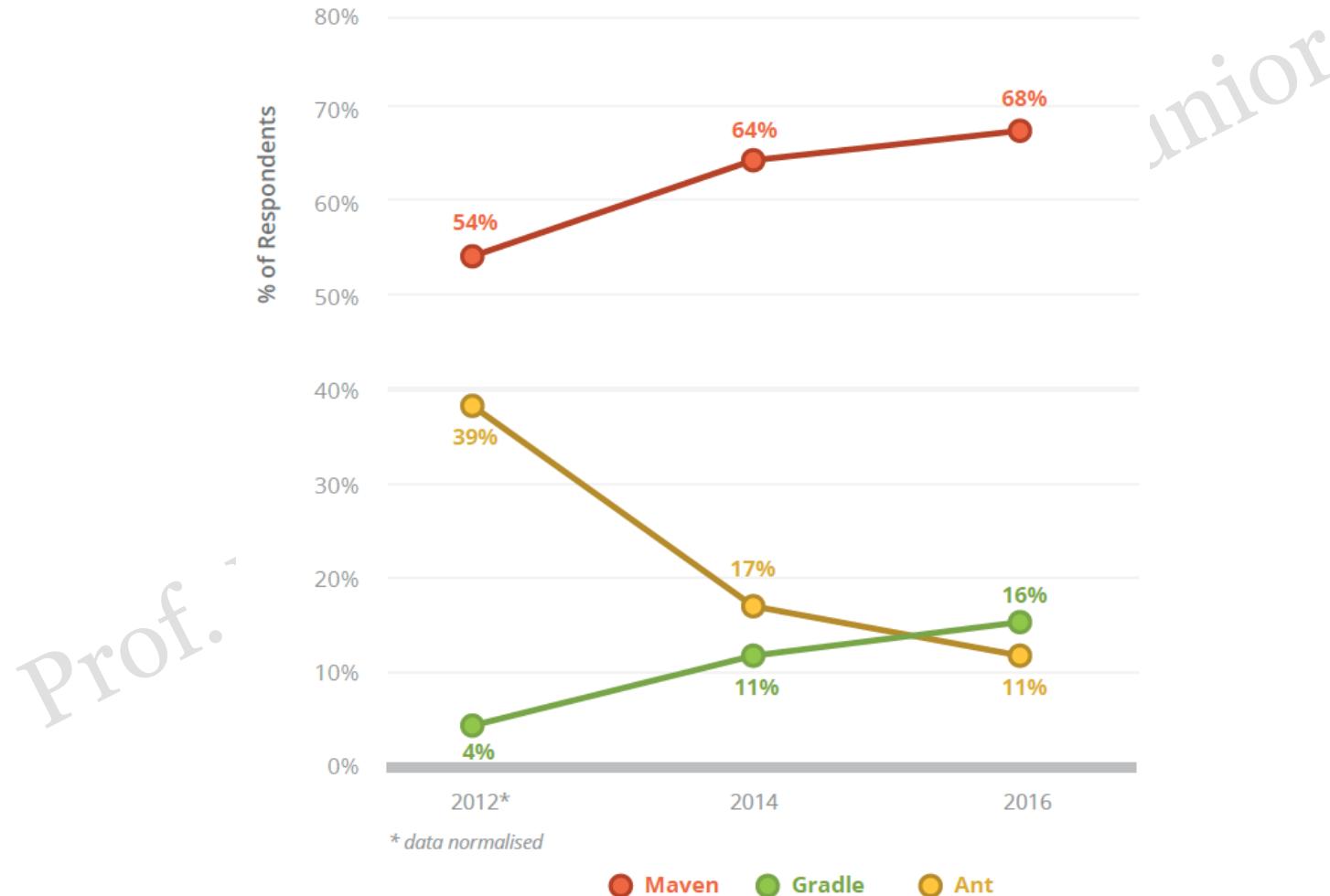


Survey sobre Ferramentas e Tecnologias Java



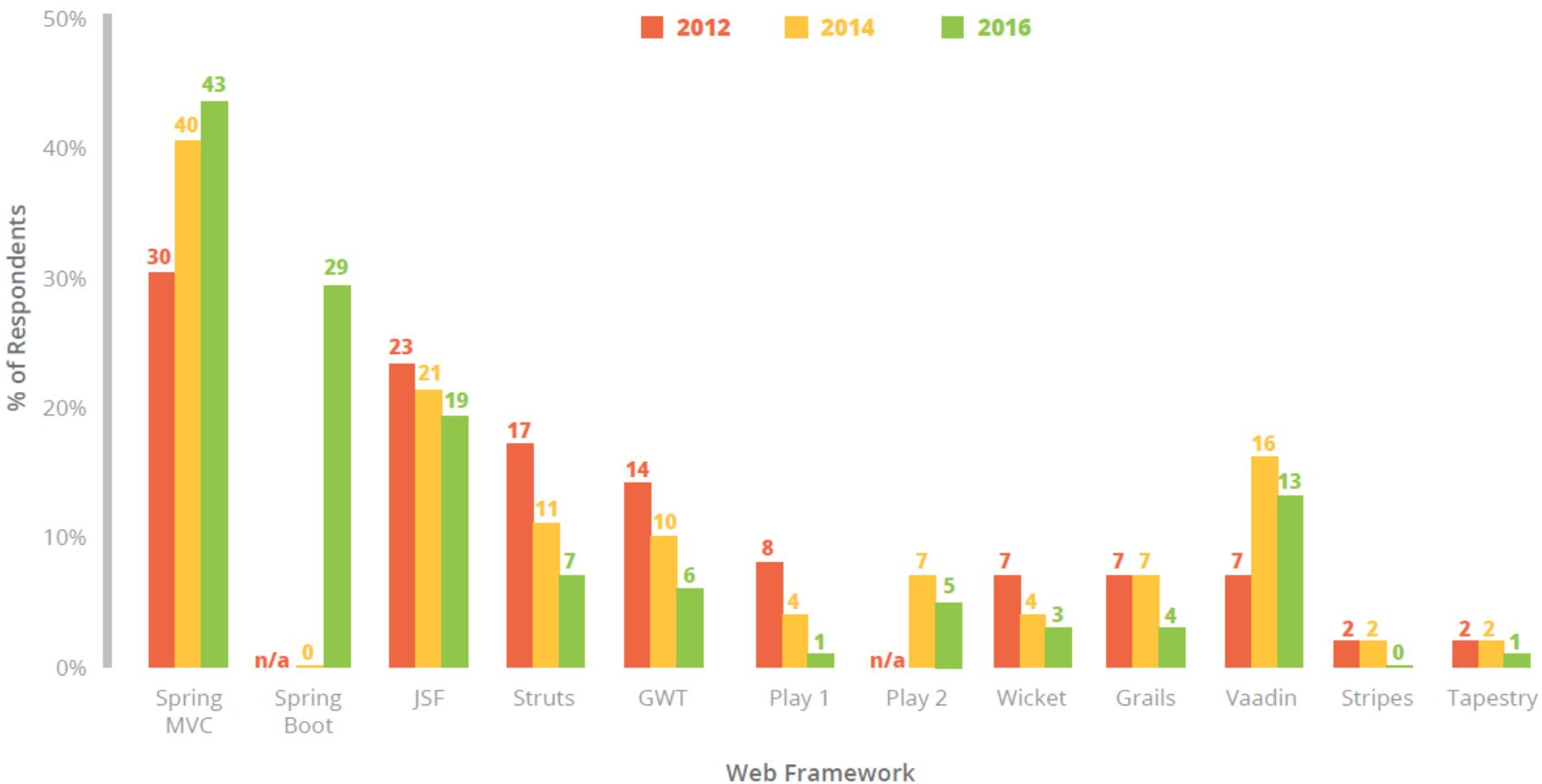


Survey sobre Ferramentas e Tecnologias Java



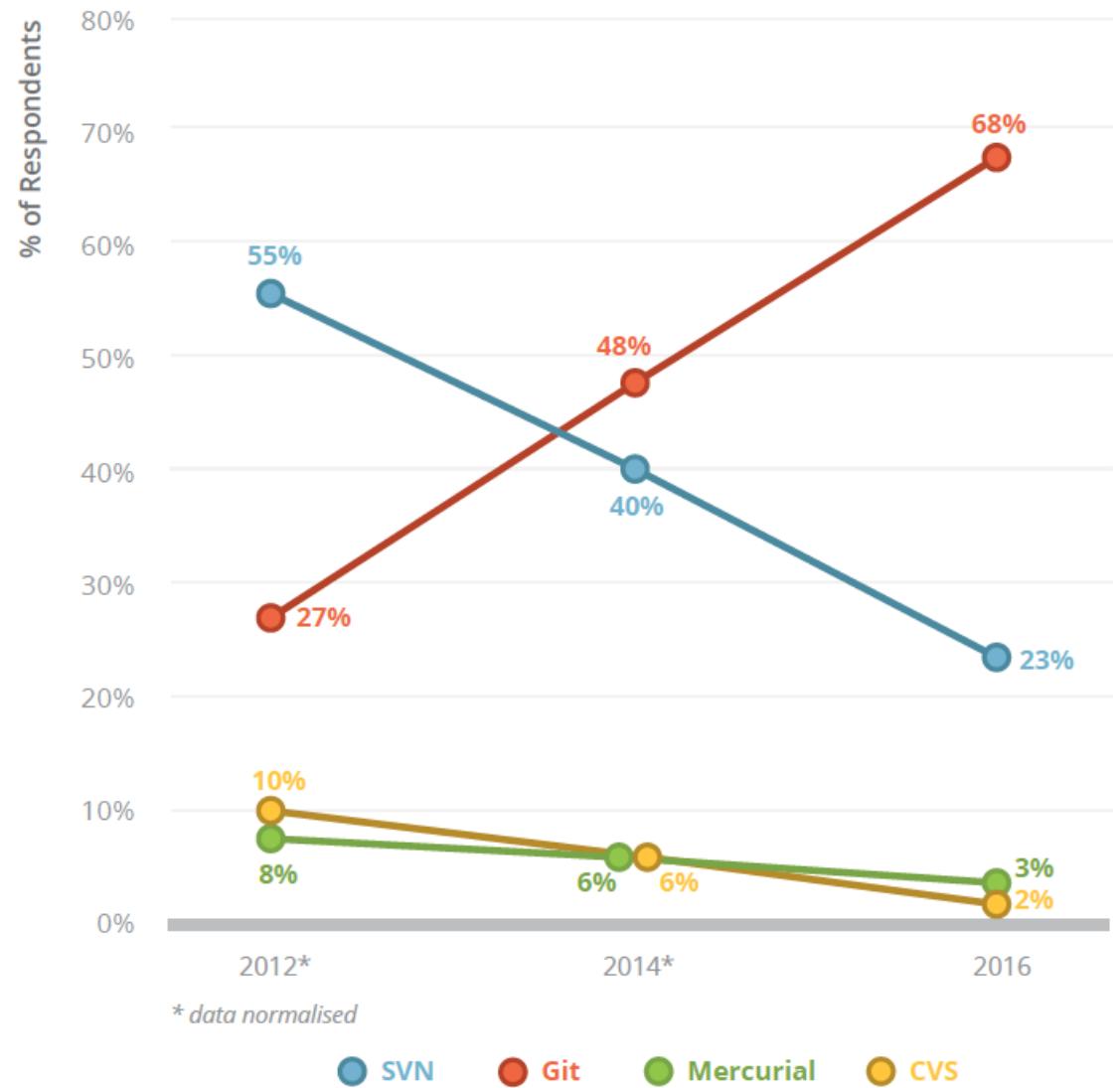


Survey sobre Ferramentas e Tecnologias Java





Survey sobre Ferramentas e Tecnologias Java





Cheat Sheet – JVM Options

Standard Options

\$ java

List all standard options.

-Dblog=RebelLabs

Sets a 'blog' system property to 'RebelLabs'. Retrieve/set it during runtime like this:

```
System.getProperty("blog");  
//RebelLabs
```

```
System.setProperty("blog", "RL");
```

-javaagent:/path/to/agent.jar
Loads the java agent in agent.jar.

-agentpath:pathname
Loads the native agent library specified by the absolute path name.

-verbose:[class/gc/jni]
Displays information about each loaded class/gc event/JNI activity.



Non-Standard Options

\$ java -X

List all non-standard options.

-Xint

Runs the application in interpreted-only mode.

-Xbootclasspath:path

Path and archive list of boot class files.

-Xloggc:filename

Log verbose GC events to filename.

-Xms1g

Set the initial size (in bytes) of the heap.

-Xmx8g

Specifies the max size (in bytes) of the heap.

-XnoClassGC

Disables class garbage collection.

-Xprof

Profiles the running program.

Advanced Options

BEHAVIOR

-XX:+UseConcMarkSweepGC

Enables CMS garbage collection.

-XX:+UseParallelGC

Enables parallel garbage collection.

-XX:+UseSerialGC

Enables serial garbage collection.

-XX:hashCode=

Modifies the results of Object.hashCode.

-XX:+FlightRecorder (requires

-XX:+UnlockCommercialFeatures)

Enables the use of the Java Flight Recorder.

DEBUGGING

-XX:ErrorFile=file.log

Save the error data to file.log.

-XX:+HeapDumpOnOutOfMemory

Enables heap dump when **OutOfMemoryError** is thrown.

-XX:+PrintGC

Enables printing messages during garbage collection.

-XX:+TraceClassLoading

Enables Trace loading of classes.

-XX:+PrintClassHistogram

Enables printing of a class instance histogram after a **Control+C** event (**SIGTERM**).

PERFORMANCE

-XX:MaxPermSize=512k (Java 7 or earlier)

Sets the max perm space size (in bytes).

-XX:ThreadStackSize=256k

Sets Thread Stack Size (in bytes).

-XX:+UseStringCache

Enables caching of commonly allocated strings.

-XX:G1HeapRegionSize=4m

Sets the sub-division size of G1 heap (in bytes).

-XX:MaxGCPauseMillis=n

Sets a target for the maximum GC pause time.

-XX:MaxNewSize=256m

Max size of new generation (in bytes).

-XX:+AggressiveOpts

Enables the use of aggressive performance optimization features.

-XX:OnError=<cmd args>

Run user-defined commands on fatal error.



Cheat Sheet – Git

Create a Repository

From scratch -- Create a new local repository

```
$ git init [project name]
```

Download from an existing repository

```
$ git clone my_url
```

Observe your Repository

List new or modified files not yet committed

```
$ git status
```

Show the changes to files not yet staged

```
$ git diff
```

Show the changes to staged files

```
$ git diff --cached
```

Show all staged and unstaged file changes

```
$ git diff HEAD
```

Show the changes between two commit ids

```
$ git diff commit1 commit2
```

List the change dates and authors for a file

```
$ git blame [file]
```

Show the file changes for a commit id and/or file

```
$ git show [commit]:[file]
```

Show full change history

```
$ git log
```

Show change history for file/directory including diffs

```
$ git log -p [file/directory]
```

Working with Branches

List all local branches

```
$ git branch
```

List all branches, local and remote

```
$ git branch -av
```

Switch to a branch, my_branch, and update working directory

```
$ git checkout my_branch
```

Create a new branch called new_branch

```
$ git branch new_branch
```

Delete the branch called my_branch

```
$ git branch -d my_branch
```

Merge branch_a into branch_b

```
$ git checkout branch_b
```

```
$ git merge branch_a
```

Tag the current commit

```
$ git tag my_tag
```

Make a change

Stages the file, ready for commit

```
$ git add [file]
```

Stage all changed files, ready for commit

```
$ git add .
```

Commit all staged files to versioned history

```
$ git commit -m "commit message"
```

Commit all your tracked files to versioned history

```
$git commit -am "commit message"
```

Unstages file, keeping the file changes

```
$ git reset [file]
```

Revert everything to the last commit

```
$ git reset --hard
```

Synchronize

Get the latest changes from origin (no merge)

```
$ git fetch
```

Fetch the latest changes from origin and merge

```
$ git pull
```

Fetch the latest changes from origin and rebase

```
$ git pull --rebase
```

Push local changes to the origin

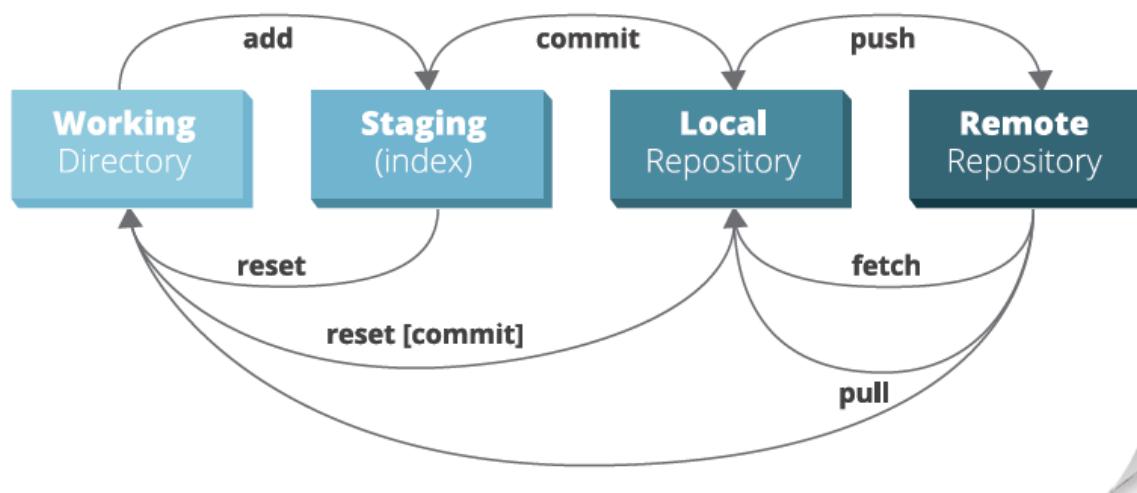
```
$ git push
```

Finally!

When in doubt, use git help

```
$ git command --help
```

Or visit <https://training.github.com/> for official GitHub training.





Cheat Sheet – Java Collections

Notable Java collections libraries

Fastutil

<http://fastutil.di.unimi.it/>

Fast & compact type-specific collections for Java
Great default choice for collections of primitive types, like int or long. Also handles big collections with more than 2^{31} elements well.

Guava

<https://github.com/google/guava>

Google Core Libraries for Java 6+
Perhaps the default collection library for Java projects. Contains a magnitude of convenient methods for creating collection, like fluent builders, as well as advanced collection types.

Eclipse Collections

<https://www.eclipse.org/collections/>

Features you want with the collections you need
Previously known as gs-collections, this library includes almost any collection you might need: primitive type collections, multimaps, bidirectional maps and so on.

JCTools

<https://github.com/JCTools/JCTools>

Java Concurrency Tools for the JVM.
If you work on high throughput concurrent applications and need a way to increase your performance, check out JCTools.

What can your collection do for you?													
Collection class	Thread-safe alternative	Your data				Operations on your collections							
		Individual elements	Key-value pairs	Duplicate element support	Primitive support	FIFO	Sorted	LIFO	Performant 'contains' check	By key	By value	By index	
HashMap	ConcurrentHashMap	✗	✓	✗	✗	✗	✗	✗	✓	✓	✗	✗	
HashBiMap (Guava)	Maps.synchronizedBiMap (new HashBiMap())	✗	✓	✗	✗	✗	✗	✗	✓	✓	✓	✗	
ArrayListMultimap (Guava)	Maps.synchronizedMultiMap (new ArrayListMultimap())	✗	✓	✓	✗	✗	✗	✗	✓	✓	✓	✗	
LinkedHashMap	Collections.synchronizedMap (new LinkedHashMap())	✗	✓	✗	✗	✓	✗	✗	✓	✓	✓	✗	
TreeMap	ConcurrentSkipListMap	✗	✓	✗	✗	✗	✓	✗	✓ * ✗ *	✓ * ✗	✗	✗	
Int2IntMap (Fastutil)		✗	✓	✗	✓	✗	✗	✗	✓	✓	✗	✓	
ArrayList	CopyOnWriteArrayList	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗	✓	
HashSet	Collections.newSetFromMap (new ConcurrentHashMap<>())	✓	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗	
ArrayList (Fastutil)		✓	✗	✓	✓	✓	✓	✓	✓	✗	✗	✓	
PriorityQueue	PriorityBlockingQueue	✓	✗	✓	✗	✗	✓ **	✗	✗	✗	✗	✗	
ArrayDeque	ArrayBlockingQueue	✓	✗	✓	✗	✓ **	✗	✓ **	✗	✗	✗	✗	

* $O(\log(n))$ complexity, while all others are $O(1)$ - constant time

** when using Queue interface methods: offer() / poll()

How fast are your collections?

Collection class	Random access by index / key	Search / Contains	Insert
ArrayList	$O(1)$	$O(n)$	$O(n)$
HashSet	$O(1)$	$O(1)$	$O(1)$
HashMap	$O(1)$	$O(1)$	$O(1)$
TreeMap	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$

Remember, not all operations are equally fast. Here's a reminder of how to treat the Big-O complexity notation:

O(1) - constant time, really fast, doesn't depend on the size of your collection

O(log(n)) - pretty fast, your collection size has to be extreme to notice a performance impact

O(n) - linear to your collection size: the larger your collection is, the slower your operations will be



Cheat Sheet – Java Generics

Basics

Generics don't exist at runtime!

```
class Pair<T1, T2> { /* ... */ }
-- the type parameter section, in angle
brackets, specifies type variables.
```

Type parameters are substituted when objects are instantiated.

```
Pair<String, Long> p1 = new
    Pair<String, Long> ("RL", 43L);
```

Avoid verbosity with the diamond operator:

```
Pair<String, Long> p1 =
    new Pair<>("RL", 43L);
```

Wildcards

`Collection<Object>` - heterogenous, any object goes in.

`Collection<?>` - homogenous collection of arbitrary type.

Avoid using wildcards in return types!

Intersection types

```
<T extends Object &
    Comparable<? super T>> T
max(Collection<? extends T> coll)
```

The return type here is **Object**!

Compiler generates the bytecode for the most general method only.

Producer Extends Consumer Super (PECS)

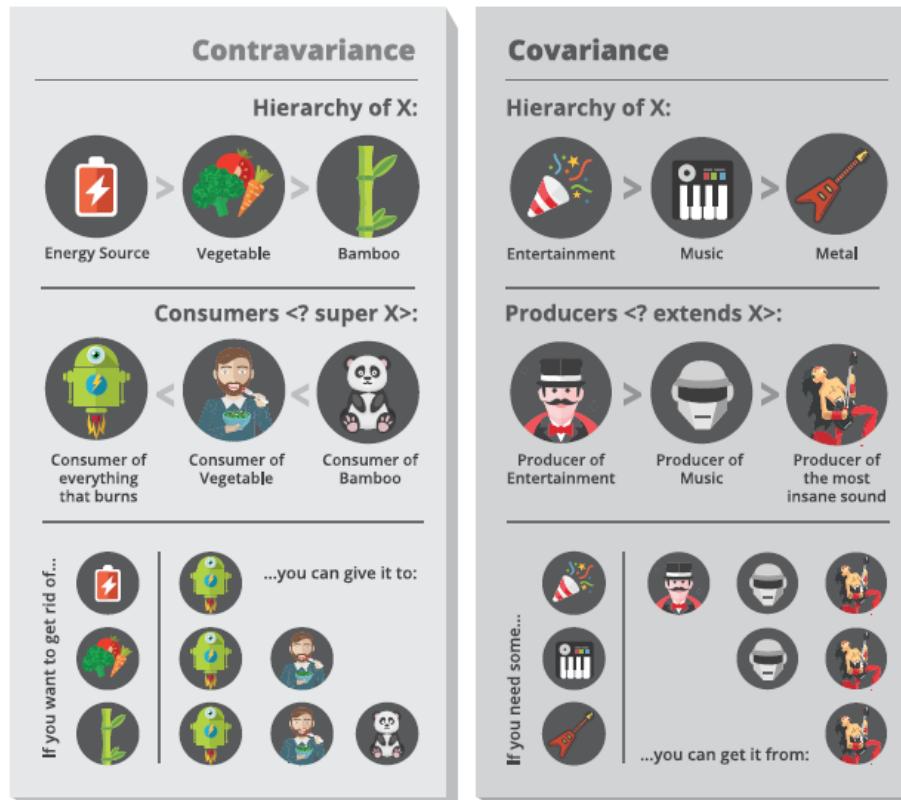
```
Collections.copy(List<? super T> dest, List<? extends T> src)
```

src -- contains elements of type T or its subtypes.

dest -- accepts elements, so defined to use T or its supertypes.

Consumers are **contravariant** (use `super`).

Producers are **covariant** (use `extends`).



Method Overloading

```
String f(Object s) {
    return "object";
}
String f(String s) {
    return "string";
}
<T> String generic(T t) {
    return f(t);
}
```

If called `generic("string")` returns "object".

Recursive generics

Recursive generics add constraints to your type variables. This helps the compiler to better understand your types and API.

```
interface Cloneable<T extends
Cloneable<T>> {
    T clone();
}
```

Now `cloneable.clone().clone()` will compile.

Covariance

```
List<Number> > ArrayList<Integer>
```

Collections are not covariant!





Cheat Sheet – Java 8 Best Practices



Default methods

Evolve interfaces & create traits

```
// Default methods in interfaces
@FunctionalInterface
interface Utilities {
    default Consumer<Runnable> m() {
        return (r) -> r.run();
    }
    // default methods, still functional
    Object function(Object o);
}

class A implements Utilities { // implement
    public Object function(Object o) {
        return new Object();
    }
    // call a default method
    Consumer<Runnable> n = new A().m();
}
}
```

Traits: 1 default method per interface
Don't enhance functional interfaces
Only conservative implementations

Lambdas

Syntax:

(parameters) -> expression
(parameters) -> { statements; }

```
// takes a Long, returns a String
Function<Long, String> f = (l) -> l.toString();
// takes nothing gives you Threads
Supplier<Thread> s = Thread::currentThread;
// takes a string as the parameter
Consumer<String> c = System.out::println;

// use them with streams
new ArrayList<String>().stream().
// peek: debug streams without changes
peek(e -> System.out.println(e)).
// map: convert every element into something
map(e -> e.hashCode()).
// filter: pass some elements through
filter(hc -> (hc % 2) == 0).
// collect all values from the stream
collect(Collectors.toCollection(TreeSet::new))
```

Rules of Thumb

Expressions over statements
Refactor to use method references
Chain lambdas rather than growing them

java.util. Optional

A container for possible null values

```
// Create an optional
Optional<String> optional =
Optional.ofNullable(a);
// process the optional
optional.map(s -> "RebelLabs:" + s);
// map a function that returns Optional
optional.flatMap(s -> Optional.ofNullable(s));

// run if the value is there
optional.ifPresent(System.out::println);
// get the value or throw an exception
optional.get();

// return the value or the given value
optional.orElse("Hello world!");
// return empty Optional if not satisfied
optional.filter(s -> s.startsWith("RebelLabs"));
```

Fields - use plain objects

Method parameters, use plain objects

Return values - use Optional

Use orElse() instead of get()



Cheat Sheet – Java 8 Streams

Definitions

- A stream **is** a pipeline of functions that can be evaluated.
- Streams **can** transform data.
- A stream **is not** a data structure.
- Streams **cannot** mutate data.

Intermediate operations

- Always return streams.
- Lazily executed.

Common examples include:

Function	Preserves count	Preserves type	Preserves order
map	✓	✗	✓
filter	✗	✓	✓
distinct	✗	✓	✓
sorted	✓	✓	✗
peek	✓	✓	✓

Stream examples

Get the unique surnames in uppercase of the first 15 book authors that are 50 years old or over.

```
library.stream()
    .map(book -> book.getAuthor())
    .filter(author -> author.getAge() >= 50)
    .distinct()
    .limit(15)
    .map(Author::getSurname)
    .map(String::toUpperCase)
    .collect(toList());
```

Compute the sum of ages of all female authors younger than 25.

```
library.stream()
    .map(Book::getAuthor)
    .filter(a -> a.getGender() == Gender.FEMALE)
    .map(Author::getAge)
    .filter(age -> age < 25)
    .reduce(0, Integer::sum);
```

Terminal operations

- Return concrete types or produce a side effect.
- Eagerly executed.

Common examples include:

Function	Output	When to use
reduce	concrete type	to cumulate elements
collect	list, map or set	to group elements
forEach	side effect	to perform a side effect on elements

Parallel streams

Parallel streams use the common ForkJoinPool for threading.

```
library.parallelStream() ...
```

or intermediate operation:

```
IntStream.range(1, 10).parallel() ...
```

Useful operations

Grouping:

```
library.stream().collect(
    groupingBy(Book::getGenre));
```

Stream ranges:

```
IntStream.range(0, 20) ...
```

Infinite streams:

```
IntStream.iterate(0, e -> e + 1) ...
```

Max/Min:

```
IntStream.range(1, 10).max();
```

FlatMap:

```
twitterList.stream()
    .map(member -> member.getFollowers())
    .flatMap(followers -> followers.stream())
    .collect(toList());
```

Pitfalls

- Don't update shared mutable variables i.e.

```
List<Book> myList =
    new ArrayList<>();
library.stream().forEach
    (e -> myList.add(e));
```

- Avoid blocking operations when using parallel streams.



Cheat Sheet – Java 8 Spring Annotations

Spring Boot and Web annotations

Use annotations to configure your web application.

T **@SpringBootApplication** - uses @Configuration, @EnableAutoConfiguration and @ComponentScan.

T **@EnableAutoConfiguration** - make Spring guess the configuration based on the classpath.

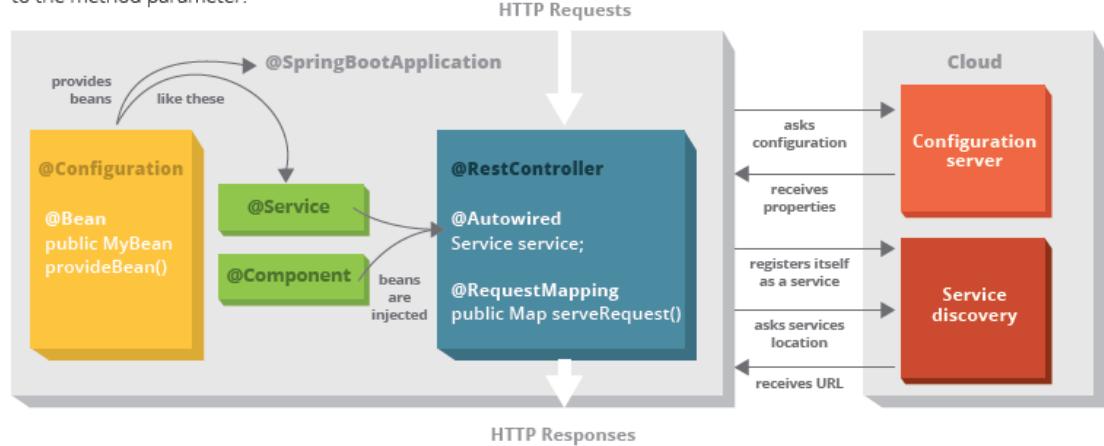
T **@Controller** - marks the class as web controller, capable of handling the requests. **T** **@RestController** - a convenience annotation of a @Controller and @ResponseBody.

M T **@ResponseBody** - makes Spring bind method's return value to the web response body.

M **@RequestMapping** - specify on the method in the controller, to map a HTTP request to the URL to this method.

P **@RequestParam** - bind HTTP parameters into method arguments.

P **@PathVariable** - binds placeholder from the URI to the method parameter.



Spring Cloud annotations

Make your application work well in the cloud.

T **@EnableConfigServer** - turns your application into a server other apps can get their configuration from.

Use **spring.application.cloud.config.uri** in the client **@SpringBootApplication** to point to the config server.

T **@EnableEurekaServer** - makes your app an Eureka discovery service, other apps can locate services through it.

T **@EnableDiscoveryClient** - makes your app register in the service discovery server and discover other services through it.

T **@EnableCircuitBreaker** - configures Hystrix circuit breaker protocols.

M **@HystrixCommand(fallbackMethod = "fallbackMethodName")** - marks methods to fall back to another method if they cannot succeed normally.

Spring Framework annotations

Spring uses dependency injection to configure and bind your application together.

T **@ComponentScan** - make Spring scan the package for the @Configuration classes.

T **@Configuration** - mark a class as a source of bean definitions.

M **@Bean** - indicates that a method produces a bean to be managed by the Spring container.

T **@Component** - turns the class into a Spring bean at the auto-scan time. **T** **@Service** - specialization of the @Component, has no encapsulated state.

C F M **@Autowired** - Spring's dependency injection wires an appropriate bean into the marked class member.

T M **@Lazy** - makes @Bean or @Component be initialized on demand rather than eagerly.

C F M **@Qualifier** - filters what beans should be used to @Autowire a field or parameter.

C F M **@Value** - indicates a default value expression for the field or parameter, typically something like `"#{systemProperties.myProp}"`

C F M **@Required** - fail the configuration, if the dependency cannot be injected.

Legend

- T** - class
- F** - field annotation
- C** - constructor annotation
- M** - method
- P** - parameter

Cheat Sheet – JUnit

Assertions and assumptions

```
Use assertions to verify the behavior under test:  

Assertions.assertEquals(Object expected,  

Object actual, Supplier<String> message)  
  

// Group assertions are run all together and  

// reported together.  

Assertions.assertAll("heading",  

() -> assertTrue(true),  

() -> assertEquals("expected",  

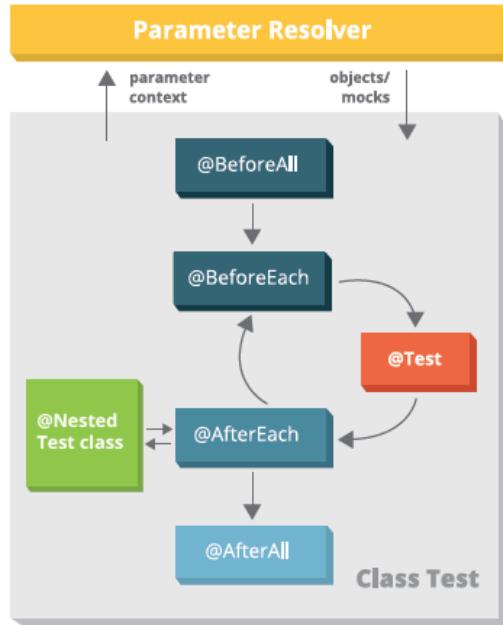
objectUnderTest.getSomething()));  
  

// To check for an exception:  

expectThrows(NullPointerException.class,  

() -> { ((Object) null).getClass();});
```

Lifecycle of standard tests



Parameter resolution

ParameterResolver - extension interface to provide parameters

```
public class MyInfoResolver implements ParameterResolver {  

    public Object resolve(ParameterContext paramCtx,  

    ExtensionContext extCtx) {  

        return new MyInfo();  

    }
}
```

Extend your tests with your parameter resolver.

```
@ExtendWith(MyInfoResolver.class)  

class MyInfoTest { ... }
```

Useful code snippets

```
@TestFactory  

Stream<DynamicTest> dynamicTests(MyContext ctx) {  

// Generates tests for every line in the file  

return Files.lines(ctx.testDataFilePath).map(l ->  

dynamicTest("Test:" + l, () -> assertTrue(runTest(l)));  

}
```

```
@ExtendWith({ MockitoExtension.class,  

DataParameterProvider.class })  

class Tests {  

    ArrayList<String> list;  
  

    @BeforeEach  

    void init() { /* init code */ }  
  

    @Test  

    @DisplayName("Add elements to ArrayList")  

void addAllToEmpty(Data dep) {  

    list.addAll(dep.getAll());  

assertAll("sizes",  

() -> assertEquals(dep.size(), list.size(),  

() -> "Unexpected size:" + instance),  

() -> assertEquals(dep.getFirst(), list.get(0),  

() -> "Wrong first element" + instance));  

}
```

Useful annotations

@Test - marks a test method

@TestFactory - method to create test cases at Runtime

@DisplayName - make reports readable with proper test names

@BeforeAll/@BeforeEach - lifecycle methods executed prior testing

@AfterAll/@AfterEach - lifecycle methods for cleanup

@Tag - declare tags to separate tests into suites

@Disabled - make JUnit skip this test.

Use @Nested on an inner class to control the order of tests.

Use @ExtendWith() to enhance the execution: provide mock parameter resolvers and specify conditional execution.

Use the lifecycle and @Test annotations on the default methods in interfaces to define contracts:

```
@Test  

interface HashCodeContract<T> {  

    <T> getValue();  

    <T> getAnotherValue();  

    @Test  

    void hashCodeConsistent() {  

        assertEquals(getValue().hashCode(),  

        getAnotherValue().hashCode(),  

        "Hashes differ");  

    }
}
```

"Never trust a test you haven't seen fail."

— Colin Vipurs



Cheat Sheet – SQL

Basic Queries

- filter your columns
SELECT col1, col2, col3, ... **FROM** table1
- filter the rows
WHERE col4 = 1 **AND** col5 = 2
- aggregate the data
GROUP by ...
- limit aggregated data
HAVING count(*) > 1
- order of the results
ORDER BY col2

Useful keywords for SELECTS:

- DISTINCT** - return unique results
BETWEEN a **AND** b - limit the range, the values can be numbers, text, or dates
LIKE - pattern search within the column text
IN (a, b, c) - check if the value is contained among given.

Data Modification

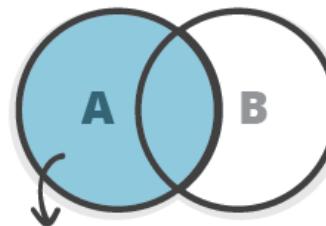
- update specific data with the **WHERE** clause
UPDATE table1 **SET** col1 = 1 **WHERE** col2 = 2
- insert values manually
INSERT INTO table1 (**ID**, **FIRST_NAME**, **LAST_NAME**)
 VALUES (1, 'Rebel', 'Labs');
- or by using the results of a query
INSERT INTO table1 (**ID**, **FIRST_NAME**, **LAST_NAME**)
 SELECT id, last_name, first_name **FROM** table2

Views

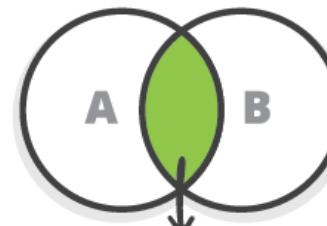
A **VIEW** is a virtual table, which is a result of a query.
They can be used to create virtual tables of complex queries.

```
CREATE VIEW view1 AS
SELECT col1, col2
FROM table1
WHERE ...
```

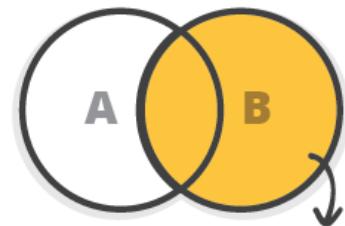
The Joy of JOINs



LEFT OUTER JOIN - all rows from table A,
even if they do not exist in table B



INNER JOIN - fetch the results that
exist in both tables



RIGHT OUTER JOIN - all rows from table B,
even if they do not exist in table A

Updates on JOINed Queries

You can use **JOINS** in your **UPDATES**
UPDATE t1 **SET** a = 1
FROM table1 t1 **JOIN** table2 t2 **ON** t1.id = t2.t1_id
WHERE t1.col1 = 0 **AND** t2.col2 **IS NULL**;

NB! Use database specific syntax, it might be faster!

Semi JOINS

You can use subqueries instead of **JOINS**:
SELECT col1, col2 **FROM** table1 **WHERE** id **IN**
 (**SELECT** t1_id **FROM** table2 **WHERE** date >
 CURRENT_TIMESTAMP)

Indexes

If you query by a column, index it!
CREATE INDEX index1 **ON** table1 (col1)

Don't forget:

- Avoid overlapping indexes
- Avoid indexing on too many columns
- Indexes can speed up **DELETE** and **UPDATE** operations

Useful Utility Functions

- convert strings to dates:
TO_DATE (Oracle, PostgreSQL), **STR_TO_DATE** (MySQL)
- return the first non-NULL argument:
COALESCE (col1, col2, "default value")
- return current time:
CURRENT_TIMESTAMP
- compute set operations on two result sets
SELECT col1, col2 **FROM** table1
UNION / EXCEPT / INTERSECT
SELECT col3, col4 **FROM** table2;

Union - returns data from both queries

Except - rows from the first query that are not present
in the second query

Intersect - rows that are returned from both queries

Reporting

Use aggregation functions

- COUNT** - return the number of rows
- SUM** - cumulate the values
- AVG** - return the average for the group
- MIN / MAX** - smallest / largest value



Universidade Estadual de Maringá (UEM)
Departamento de Informática (DIN)



Frameworks Java

Prof. Dr. Edson A. Oliveira Junior
edson@din.uem.br
www.din.uem.br/~edson



Lista de Frameworks Java - iMasters

- <https://imasters.com.br/linguagens/java/listagem-de-frameworks-java>

Prof. Dr. Edson A. Oliveira Junior
edson@din.uem.br



Universidade Estadual de Maringá (UEM)
Departamento de Informática (DIN)



Introdução ao Spring MVC

Prof. Dr. Edson A. Oliveira Junior
edson@din.uem.br
www.din.uem.br/~edson

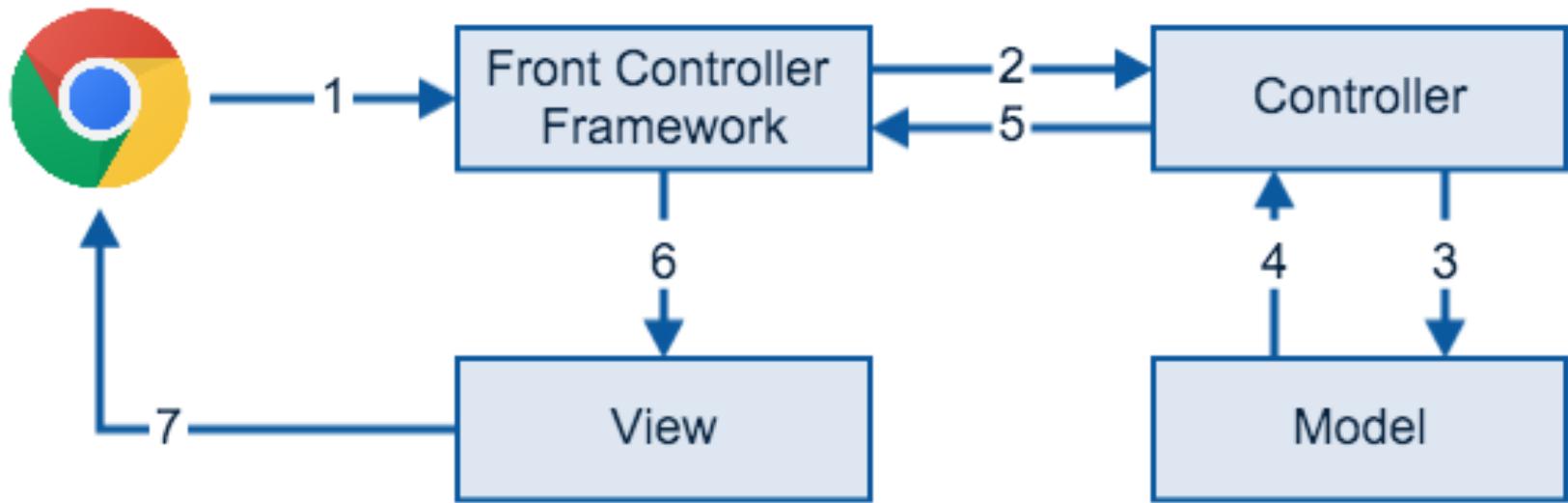


Spring MVC

- O que é o Spring MVC?
 - Framework de apoio ao desenvolvimento de aplicações Web em Java
 - Implementação do MVC do Spring Framework
- Principais funcionalidades implementadas:
 - atende requisições HTTP;
 - delega responsabilidades de processamento de dados a outros componentes;
 - prepara resposta ao cliente.

Spring MVC

- Fluxo de controle em Spring MVC



Spring MVC

- Fluxo de controle em Spring MVC:
 1. **Browser** envia requisição HTTP ao servidor com a aplicação Spring MVC;
 2. O **Front Controller** identifica a classe Controller responsável para tratar a requisição;
 3. O **Controller** envia os dados ao Model para execução de operações;
 4. O **Model** envia o resultado das operações é devolvido ao Controller;
 5. O **Controller** retorna ao Front Controller o nome da View para exibir os resultados;
 6. O **Front Controller** busca a View que processa os dados, gerando um HTML; e
 7. A **View** retorna o HTML ao browser.



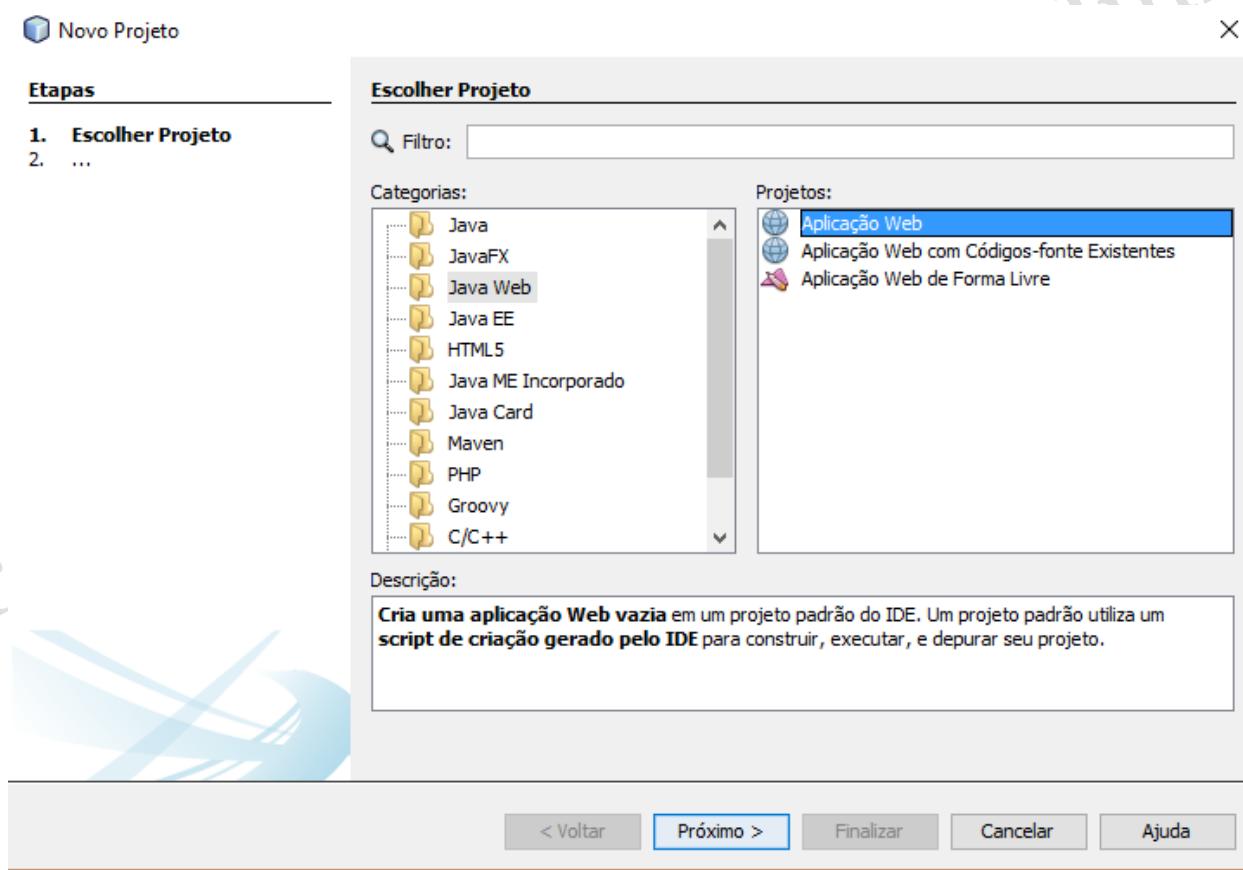
Spring MVC - Criando uma Aplicação

- Há 2 maneiras de fazer:
 - Criando uma aplicação Web comum simples;
 - Usando o Spring Boot
- Nesta aula vamos seguir a primeira opção por termos pouco tempo disponível. Porém, o mais recomendado é seguir a segunda opção!



Configurando um Projeto no Netbeans

- Vá em: Arquivo/Novo Projeto
- Em Categorias escolha: Java Web
- Em Projetos escolha: Aplicação Web
- Clique em Próximo



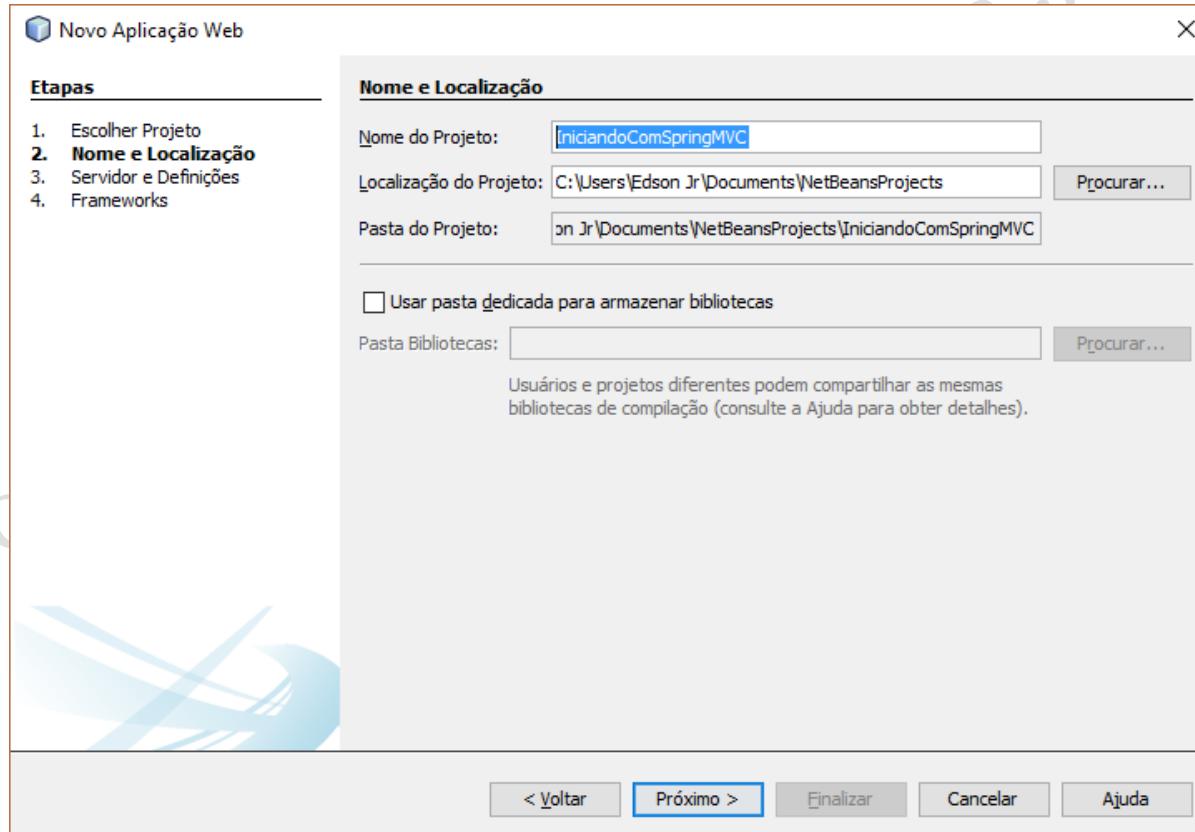
Configurando um Projeto no Netbeans

- Dê um nome ao seu projeto em

Nome do Projeto →

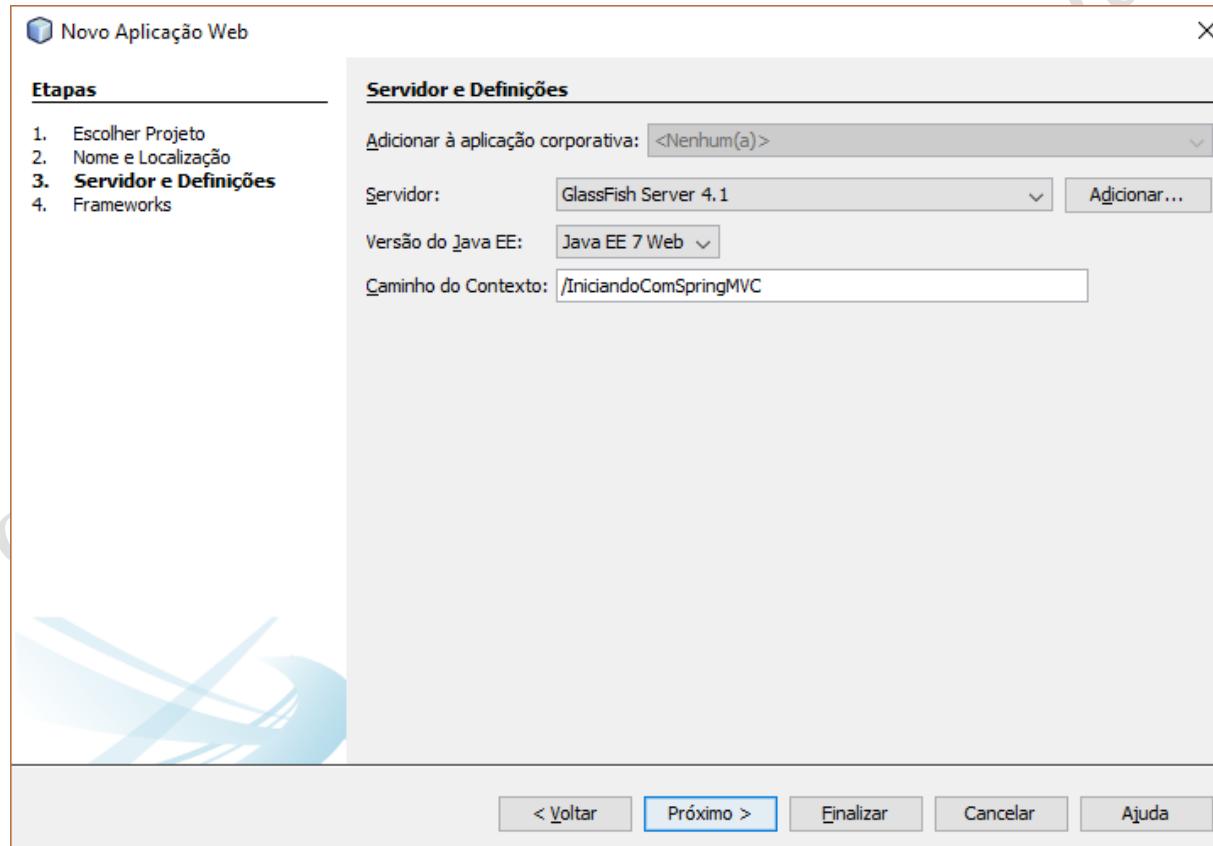
IniciandoComSpringMVC

- Clique em Próximo



Configurando um Projeto no Netbeans

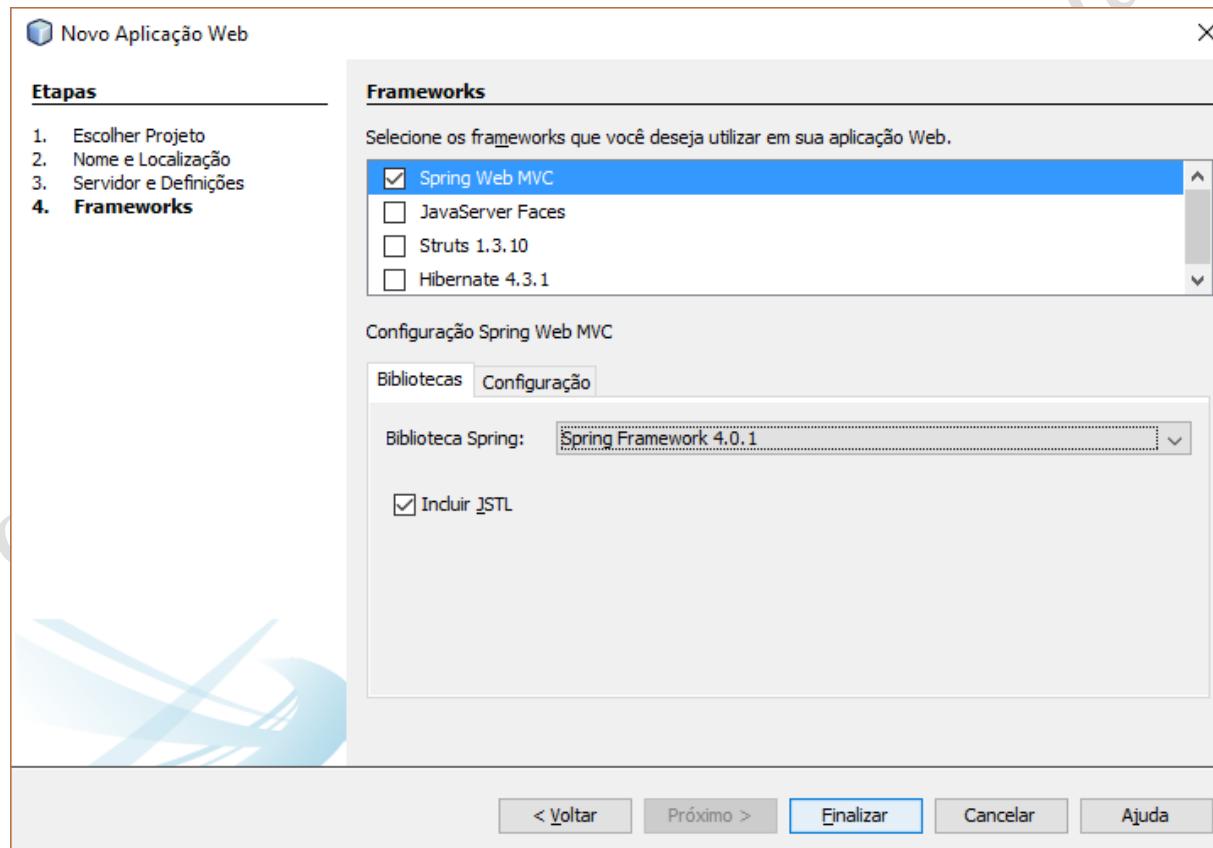
- Escolha o Servidor de sua preferência: GlassFish ou Tomcat
- Escolha a versão do Java EE
- De um nome ao contexto (root) da aplicação → /IniciandoComSpringMVC
- Clique em Próximo





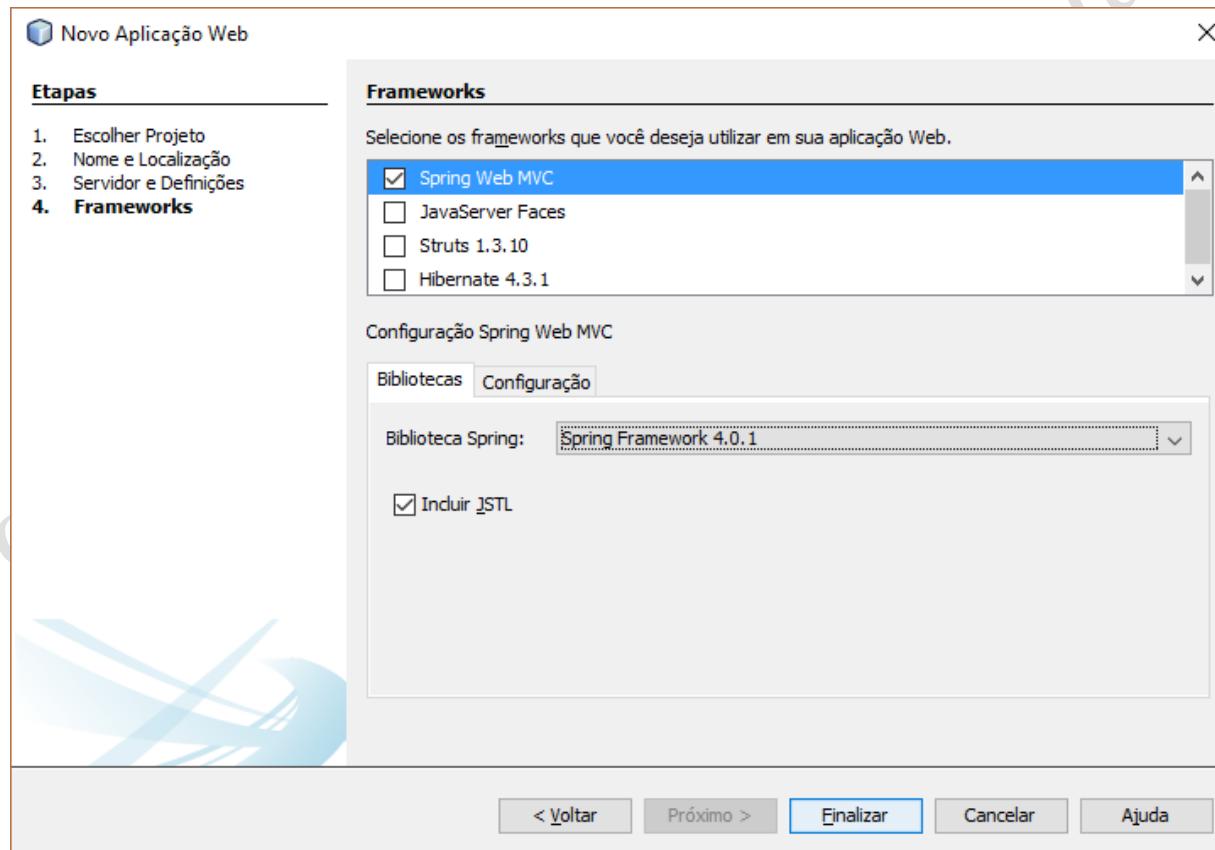
Configurando um Projeto no Netbeans

- Escolha o(s) Framework(s) que você precisa de suporte → Spring Web MVC
 - Em Bibliotecas Spring escolha a versão desejada
 - Marque Incluir JSTL



Configurando um Projeto no Netbeans

- Na aba Configuração, mude o Mapeamento de emissores para “/”
- Clique em Finalizar





Configurando um Projeto no Netbeans

- Veja a estrutura gerada pelo NB

NetBeans IDE 8.0.2

Arquivo Editar Exibir Navegar Código-Fonte Refatorar Executar Depurar Perfil Equipe Ferramentas Janela Ajuda

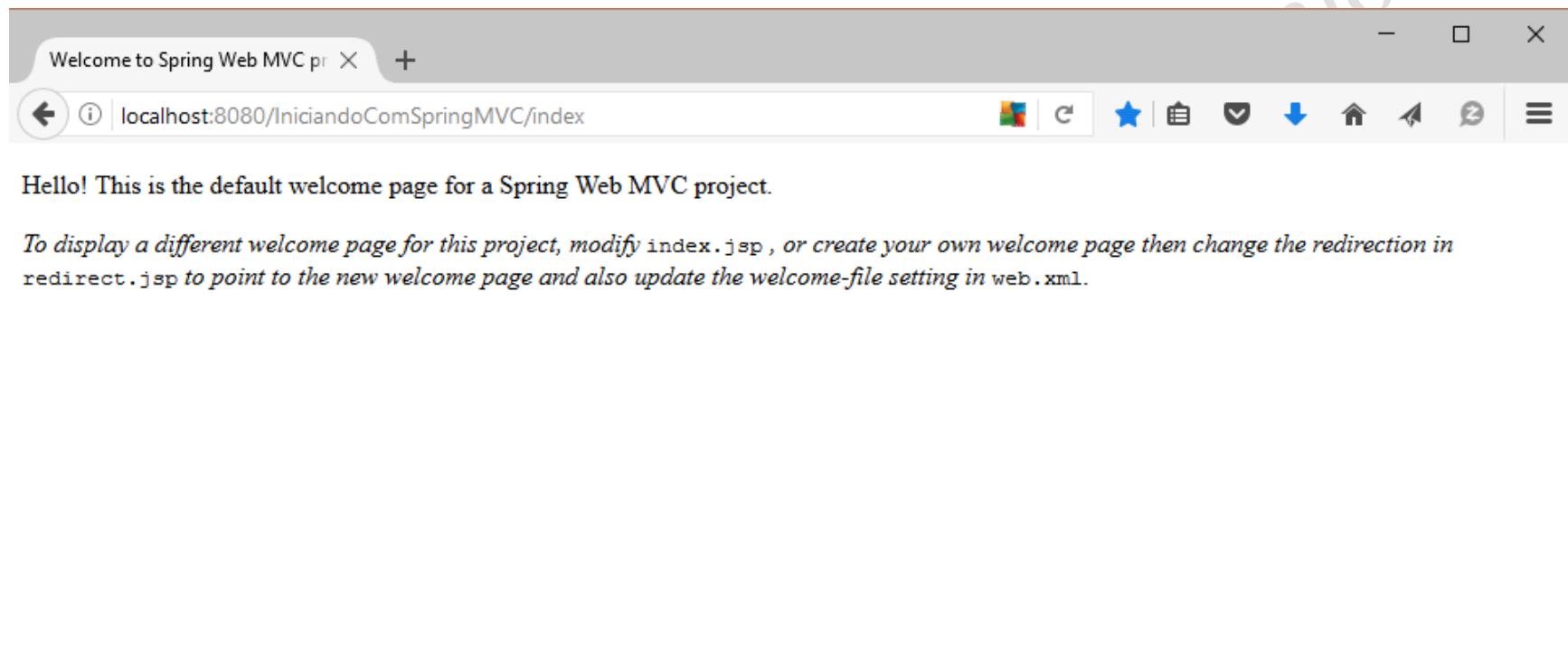
The screenshot shows the NetBeans IDE interface with the following details:

- Projects View:** Shows the project structure under "IniciandoComSpringMVC". It includes a "Páginas Web" node with "WEB-INF" containing "index.jsp", "applicationContext.xml", "dispatcher-servlet.xml", and "web.xml". There is also a "redirect.jsp" file. A "Pacotes de Códigos-fonte" node contains a "pacote default" package. A "Bibliotecas" node lists numerous Spring Framework jars (e.g., Spring Framework 4.0.1 - org.apache.commons.logging.jar, Spring Framework 4.0.1 - cglb-2.2.jar, etc.) and other dependencies like GlassFish Server 4.1, JSTL, and JDK 1.8.
- Código-Fonte View:** Displays the content of the "applicationContext.xml" file. The XML code defines a bean for a "dataSource" using "DriverManagerDataSource" and a "propertyConfigurer" using "PropertyPlaceholderConfigurer". It also includes a placeholder for persistence support with the comment "--> ADD PERSISTENCE SUPPORT HERE (jpa, hibernate, etc) -->".
- Navegador View:** Shows the "applicationContext.xml - Navegador" panel with the XML code.
- Filtros:** At the bottom left, there are filter icons for Java, XML, and CSS.



Configurando um Projeto no Netbeans

- Executando....





Configurando um Projeto no Netbeans

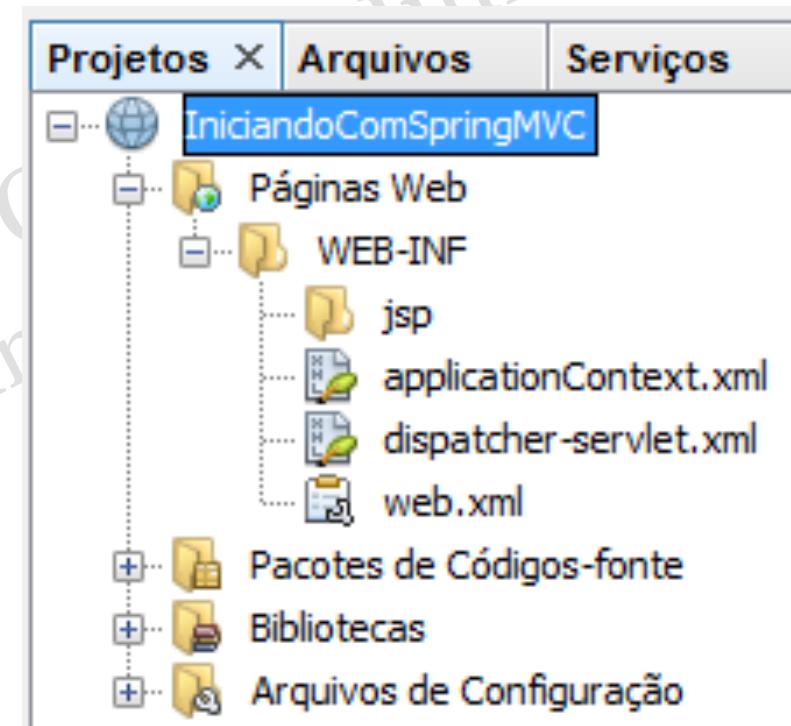
- Removendo o código gerado que é desnecessário por enquanto..

```
<?xml version='1.0' encoding='UTF-8' ?>
<!-- was: <?xml version="1.0" encoding="UTF-8"?> -->
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
        http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
        http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">
<!--
Most controllers will extend ControllerClassNameHandlerMapping
for the index controller, so we can define a general mapping + must
define an explicit mapping for the index controller
-->
<bean id="urlMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <props>
            <prop key="indexController">indexController</prop>
        </props>
    </property>
</bean>
<bean id="viewResolver"
    class="org.springframework.web.servlet.view.InternalResourceViewResolver"
    p:prefix="/WEB-INF/jsp/"
    p:suffix=".jsp" />
<!--
The index controller.
-->
<bean name="indexController"
    class="org.springframework.web.servlet.mvc.Controller"
    p:viewName="indexController" />
</beans>
```



Configurando um Projeto no Netbeans

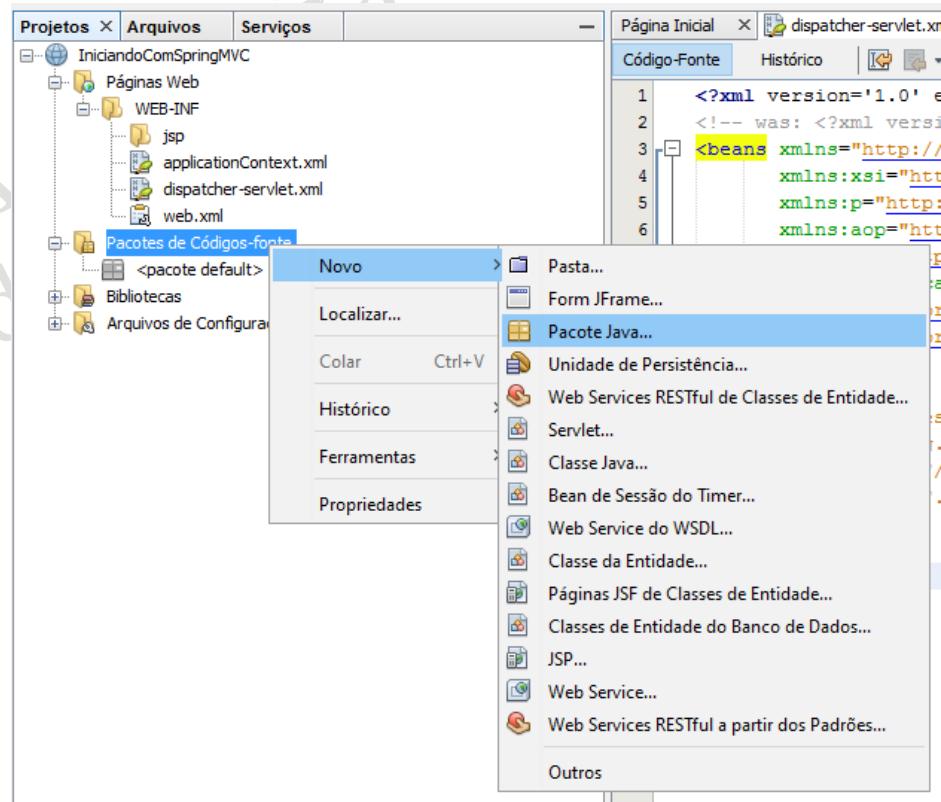
- Apagar o redirect.jsp e o index.jsp para podermos criar o nosso próprio Controller!!! Vamos trabalhar com URL Mapping somente!!!





Criando um Controller

- Todo Controller é uma classe, logo devemos criar tal classe em Pacotes de Códigos-fonte, certo?
- Vá em Pacotes de Códigos-fonte e clique com o botão direito
- Escolha a opção Novo/Pacote Java
- Dê um nome ao pacote:
 - br.uem.din.grsse.springmvc.controller





Criando um Controller

- Para a criação dos controllers usaremos Java Annotations
- Para isso, inclua o seguinte código no arquivo dispatcher-servlet.xml
 - <mvc:annotation-driven />
 - <context:component-scan base-package="br.uem.din.grsse.springmvc" />
- Em seguida, inclua as seguintes linhas na tag beans
 - xmlns:mvc="http://www.springframework.org/schema/mvc"
 - xmlns:context="http://www.springframework.org/schema/context"
 - Inclua as linhas abaixo na propriedade xsi:schemaLocation:
 - http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
 - http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd



Criando um Controller



```
<?xml version='1.0' encoding='UTF-8' ?>
<!-- was: <?xml version="1.0" encoding="UTF-8"?> -->

```



Criando um Controller

- Para criar a classe controller clique com o botão direito no pacote criado e escolha a opção Novo/Classe Java
- Dê um nome para a classe → MeuController
- Anote a classe da seguinte forma:
 - Antes da assinatura da classe inclua
 - @Controller
- Crie o método a seguir:
 - public String index() { return "index";}
 - Anote-o com @RequestMapping("/inicio")
- Veja no próximo slide!!



Criando um Controller

The screenshot shows a Java code editor with the following code:

```
1 package br.uem.din.grsse.springmvc.controller;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.RequestMapping;
5
6 /**
7 *
8 * @author Edson OliveiraJr <edson@din.uem.br> - www.din.uem.br/~edson
9 */
10
11 @Controller
12 public class MeuController {
13
14     @RequestMapping("/inicio")
15     public String index() {
16         return "index";
17     }
18 }
19 }
```

The code defines a Spring MVC controller named `MeuController`. It contains a single method `index()` mapped to the URL `/inicio`, which returns the view name `"index"`. The code is annotated with `@Controller` and `@RequestMapping`. A copyright notice is present at the top of the file.



Criando o index.jsp

- Na pasta jsp crie um arquivo do tipo JSP
- Clique com o botão direito na pasta jsp e escolha a opção Novo/JSP...
- Dê o nome de index
- Obs.: o nome do arquivo tem que ser exatamente o mesmo retorno do método criado na classe controller MeuController: "index"
- Execute o projeto...
- Acesse: <http://localhost:8080/IniciandoComSpringMVC/inicio>



Adicionando um formulário de cadastro

- Em index.jsp adicione:
 - Clique aqui para cadastrar!!
- Crie uma nova pasta em "jsp" chamada:
 - "pessoas"
 - Nesta pasta crie um jsp chamado cadastroPessoas.jsp
 - O conteúdo deste arquivo está disponível em:
 - www.din.uem.br/~edson/cadastroPessoas.jsp



Adicionando um formulário de cadastro

- Em MeuController inclua o seguinte método:

```
public String cadastro() { return "pessoas/cadastroPessoa"; }
```

- Anote o método criado com: `RequestMapping("/cadastrarAlguem")`



Adicionando um formulário de cadastro

Preencha o formulário abaixo

- Execute o projeto

- Acesse:

- [http://localhost:](http://localhost:8080/)

[8080/](#)

[IniciandoComSprin](#)

[gMVC/](#)

[cadastrarAlguem](#)

Dados Pessoais

Nome: Sobrenome:
Nascimento: dd mm aaaa
RG:
CPF: -

Dados de Endereço

Rua: Número:
Bairro:
Estado:
Cidade:
CEP: -

Dados de login

E-mail:
Imagen de perfil: Nenhum arquivo selecionado.
Login de usuário:
Senha:
Confirme a senha:



Adicionando CSS

- Em Páginas Web crie a pasta: resources/css
- Copie um arquivo CSS qualquer para a pasta css
- Inclua a seguinte tag em dispatcher-servlet.xml
 - <mvc:resources mapping="/resources/**" location="/resources/" />
- Em cadastroPessoa.jsp incluir:
 - <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
 - <link href=<c:url value="resources/css/style.css" />" rel="stylesheet" type="text/css" />
- Aplique o CSS ao cadastroPessoa.jsp



Adicionando CSS

- Execute o projeto
- Acesse:

<http://localhost:8080/>

[IniciandoComSpringMVC/](#)
[cadastrarAlguem](#)

Preencha o formulário abaixo

Dados Pessoais

Nome: Sobrenome:

Nascimento: dd mm aaaa

RG:

CPF:

Dados de Endereço

Rua: Número:

Bairro:

Estado: Acre

Cidade:

CEP:

Dados de login

E-mail:

Imagen de perfil: Selecionar arquivo... Nenhum arquivo selecionado.

Login de usuário:

Senha:

Confirme a senha:



Tarefa

- Continue trabalhando no projeto IniciandoComSpringMVC
- Crie uma aplicação que permita o cadastro de:
 - Contas bancárias (poupança ou conta corrente);
 - Vincule pelo menos uma conta a cada pessoa;
 - Forneça o relatório em tela das contas vinculadas a uma determinada pessoa;
 - Imprima em tela o extrato completo de uma conta com base no mês atual ou no mês escolhido pelo usuário;
 - Permita o saque e o depósito em qualquer conta.



Tarefa

- Instruções:
 - Toda a persistência deverá ser feita em memória primária usando Java Collections;
 - Utilize os recursos front-end que achar necessário na sua aplicação.
Ex.: JavaScript para validação, CSS para formatação, etc;
 - Cada grupo em trio;
 - Terminar em sala de aula ou trazer pronto para a próxima aula.
- Obs.: este projeto será usado na próxima aula!