



Bem-vindos

Acelera Atos



Quem é o Prof.

Sérgio Gomes

- Pós Graduado em Ciência de dados e Big data - PUC
- Pós Graduado em Arquitetura de Sistemas distribuídos - PUC
- Formado em Sistema da Informação - USJT
- Oracle JavaEE 6 - Enterprise Architect Certified Master - OCMJEA
- Sun Certified Web Component Java EE 5 SCWCD
- Sun Certified Programmer Java SE 5 - SCJP
- Atuo como Arquiteto de Soluções pela WA-BR consultoria
- Desenvolvedor desde 1996

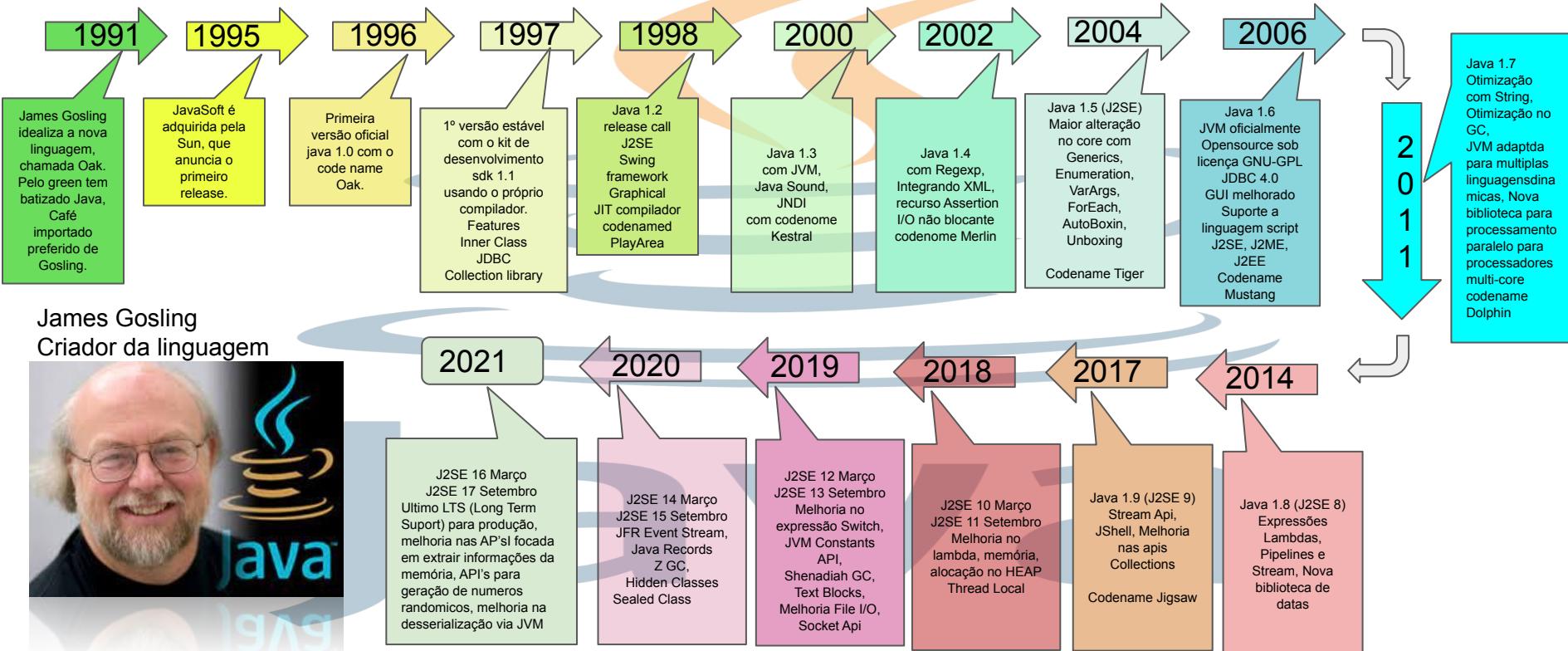
Introdução ao JAVA

01

Histórico da Linguagem



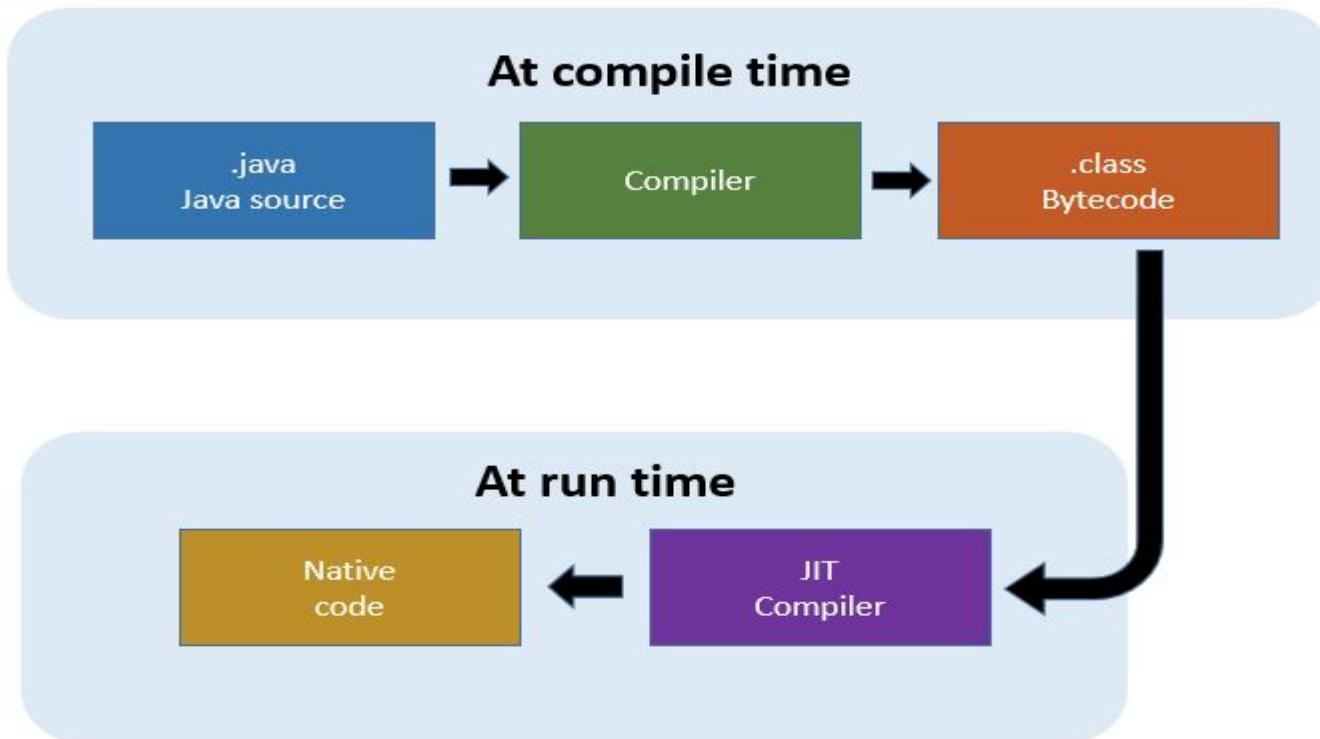
Escreva uma Vez e rode em qualquer lugar



fonte: <https://dzone.com/articles/why-java-is-so-young-after-25-years-an-architects>

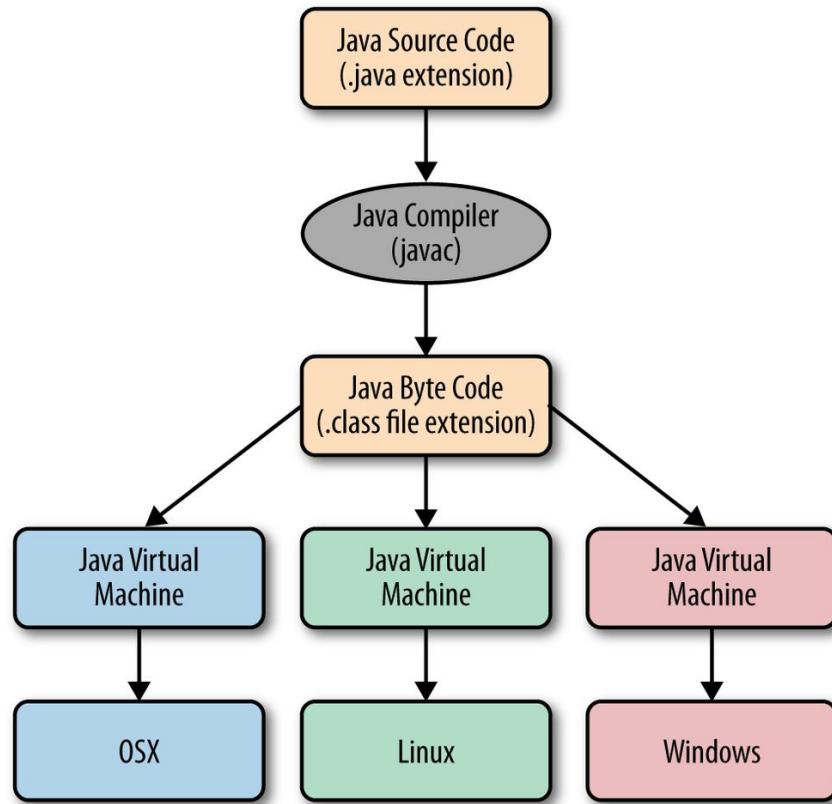


Processo de compilação e execução



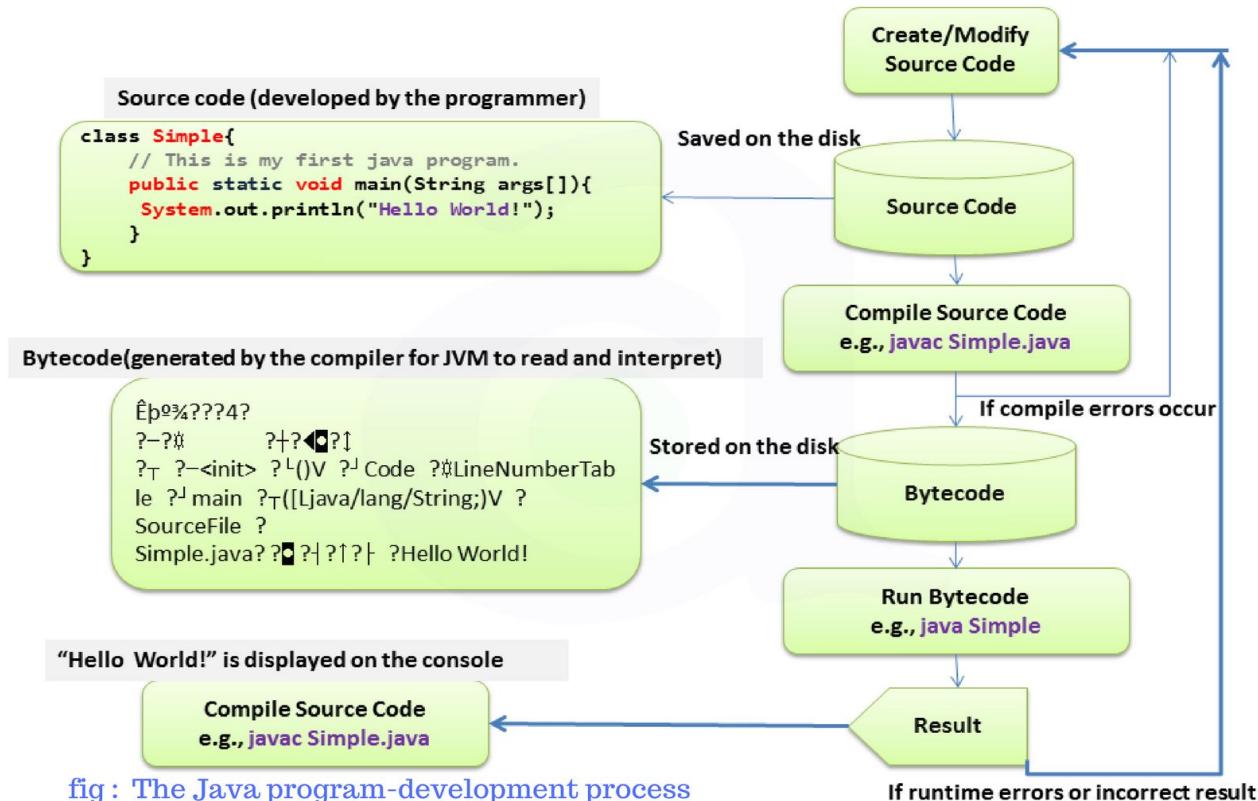


Processo de execução do programa





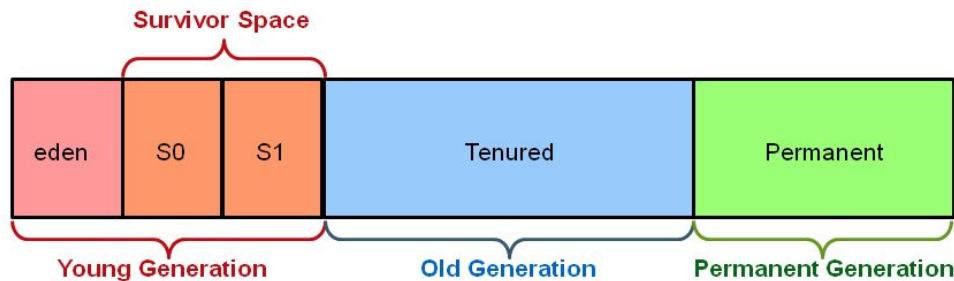
Processo de compilação e execução do programa



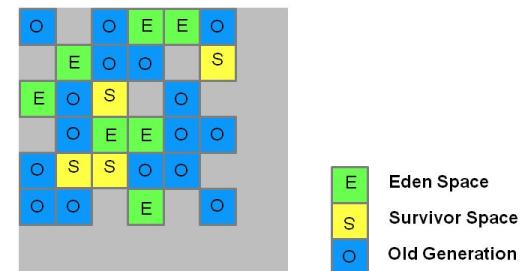


Processo de Garbage Collector

Hotspot Heap Structure



G1 Heap Allocation





Plataforma Java

Java SE

Java
Standard
Edition

Java EE

Java
Enterprise
Edition

Java ME

Java Micro
Edition

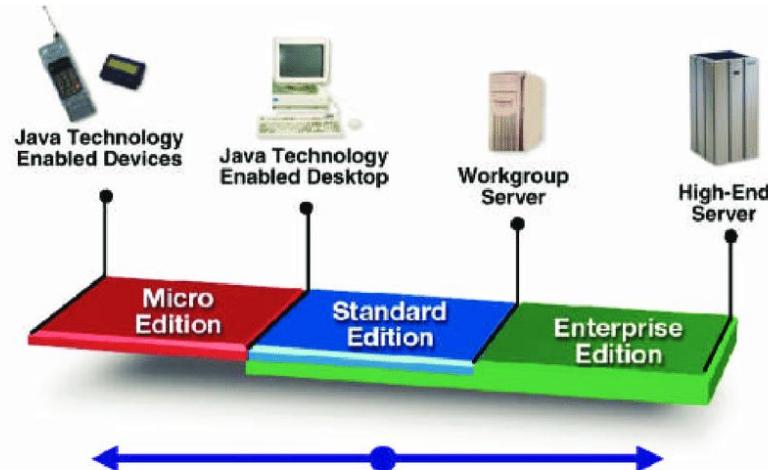
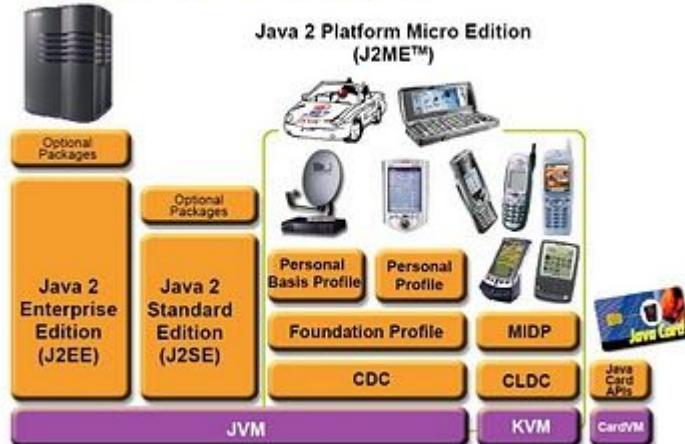
JavaFX

Plataforma
de
software
multimídia



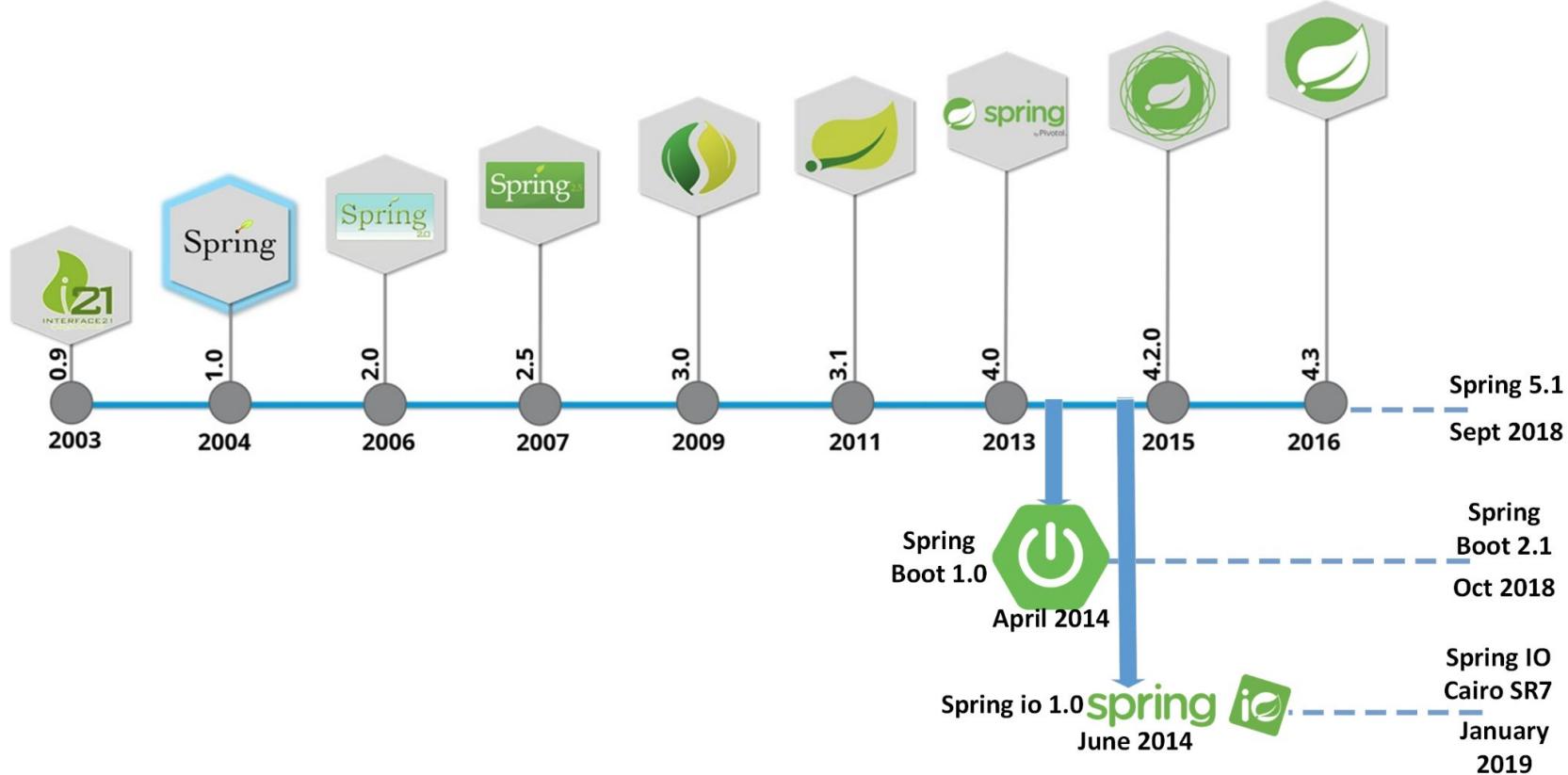
Plataforma Java

The Java™ Platform



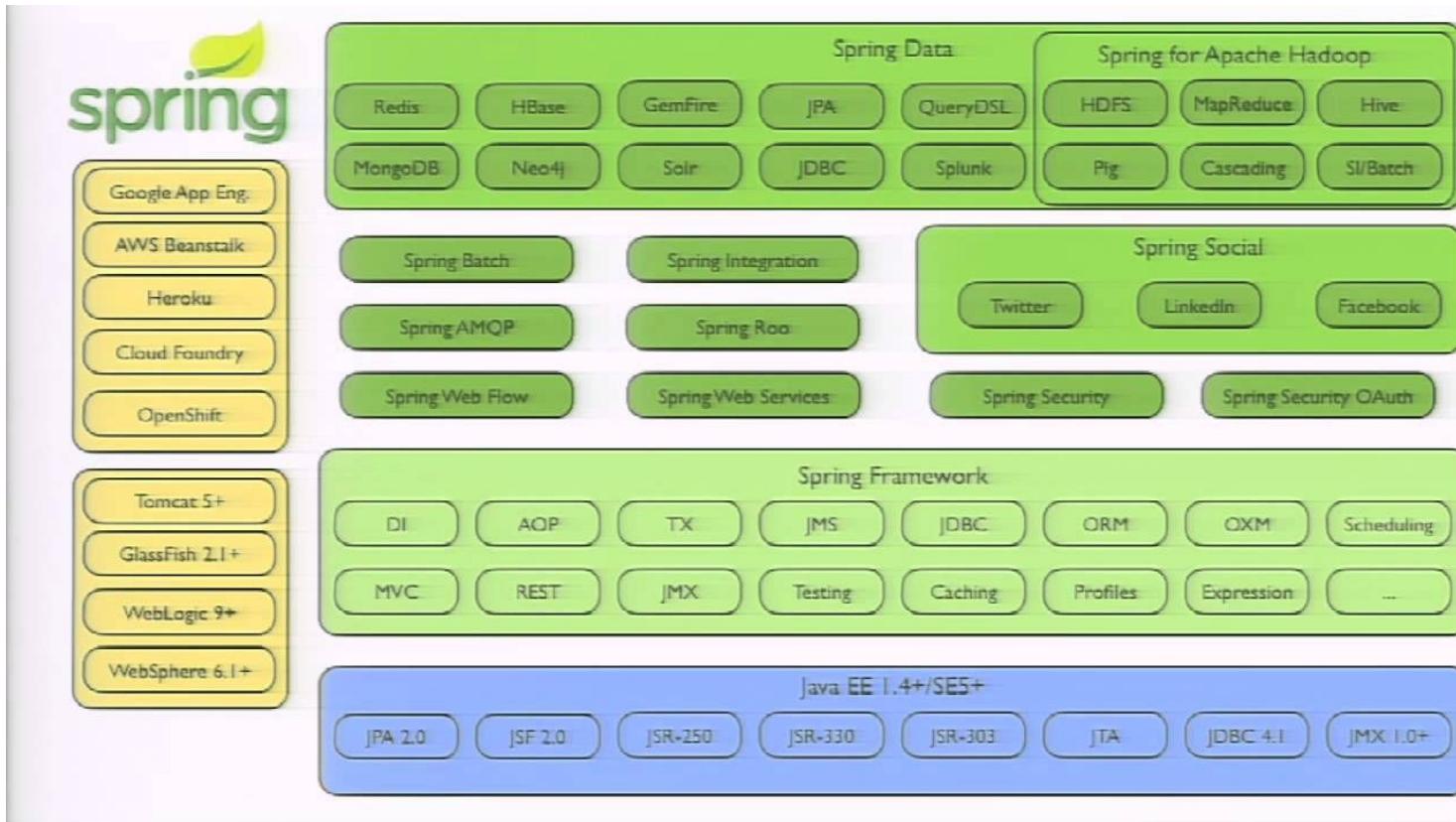


Comunidade Java e o Spring





Comunidade Java e o Spring





IDE - Integrated Development Environment

Ranked: The Big Five Java IDEs

Please don't



Simply fantastic



.....



NetBeans



Visual Studio Code



IntelliJ IDEA

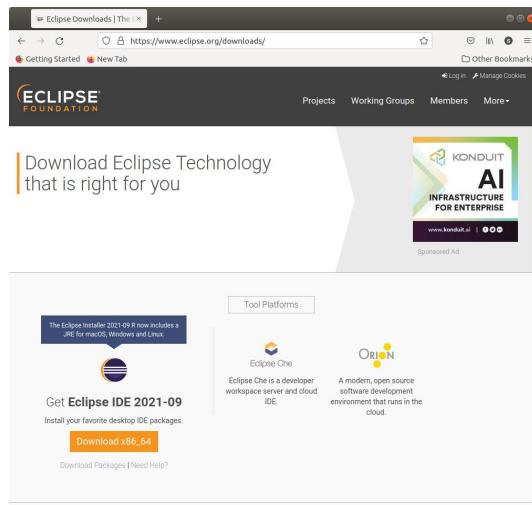


Preparando o ambiente

<https://openjdk.java.net/install/>



OpenJDK™ 11



eclipseinstaller	
	by Oomph
<input type="text" value="type filter text"/> q	
	<h3>Eclipse IDE for Java Developers</h3> <p>The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Maven and Gradle integration</p>
	<h3>Eclipse IDE for Enterprise Java and Web Developers</h3> <p>Tools for developers working with Java and Web applications, including a Java IDE, tools for JavaScript, TypeScript, JavaServer Pages and Faces, Yaml, Markdown, Web Services,...</p>
	<h3>Eclipse IDE for C/C++ Developers</h3> <p>An IDE for C/C++ developers.</p>
	<h3>Eclipse IDE for Embedded C/C++ Developers</h3> <p>An IDE for Embedded C/C++ developers. It includes managed cross build plug-ins (Arm and RISC-V) and debug plug-ins (SEGGER J-Link, OpenOCD, pyocd, and QEMU), plus ...</p>
	<h3>Eclipse IDE for PHP Developers</h3> <p>The essential tools for any PHP developer, including PHP language support, Git client, Maven and editors for JavaScript, TypeScript, HTML, CSS and XML</p>

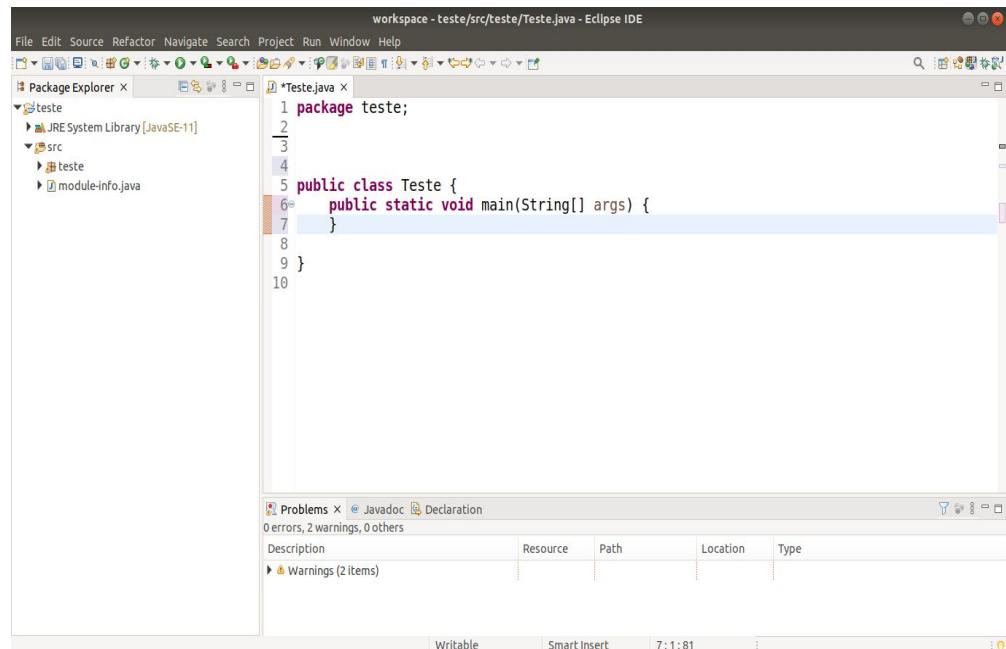
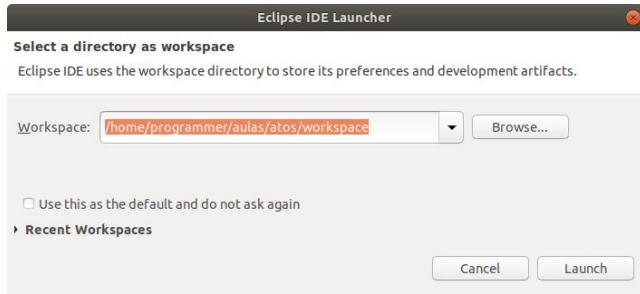
Introdução ao JAVA

01

Logica de programação em Java



O Começo





Palavras Reservadas

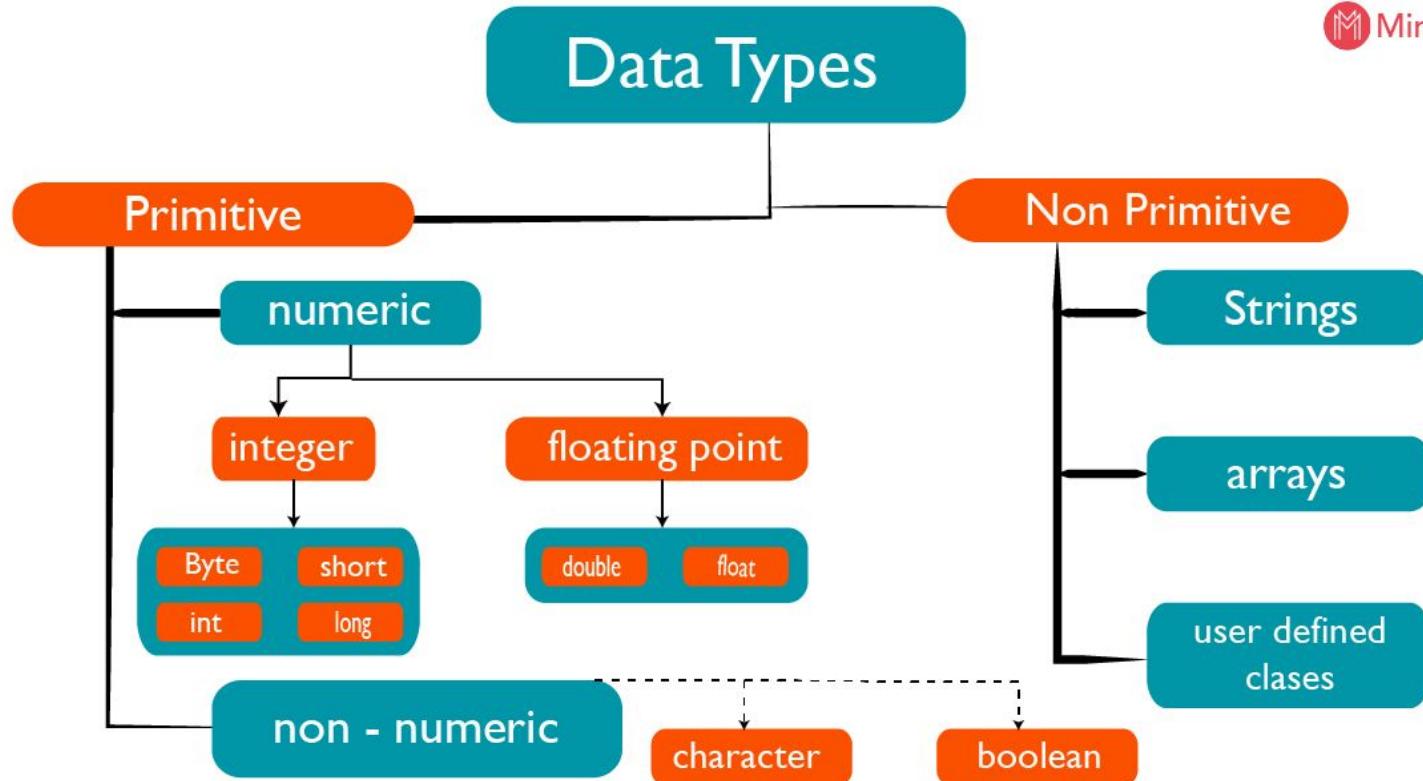
Complete List of Java Keywords

abstract	boolean	break	byte	case	catch
char	class	const	continue	default	do
double	else	extends	final	finally	float
for	goto	if	implements	import	instanceof
int	interface	long	native	new	package
private	protected	public	return	short	static
strictfp	super	switch	synchronized	this	throw
throws	transient	try	void	volatile	while
assert					



Tipo de dados

M MindMajix





Tipo de dados

Type	Size (in bits)	Range
byte	8	-128 to 127
short	16	-32,768 to 32,767
int	32	-2^{31} to $2^{31}-1$
long	64	-2^{63} to $2^{63}-1$
float	32	1.4e-045 to 3.4e+038
double	64	4.9e-324 to 1.8e+308
char	16	0 to 65,535
boolean	1	true or false



Encerramos por hoje !!!

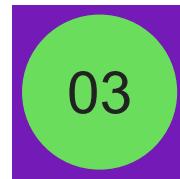




Título aqui



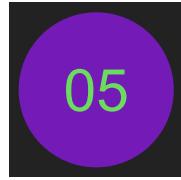
Título aqui



Título aqui

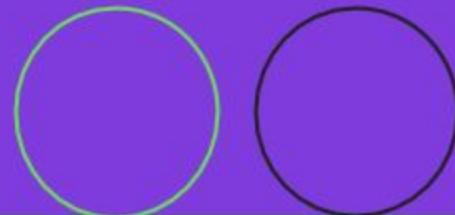


Título aqui



Título aqui

POO



Iniciando o entendimento

Classes

01

Estrutura de uma classe básica



O que é uma classe

Classe é um tipo de dado

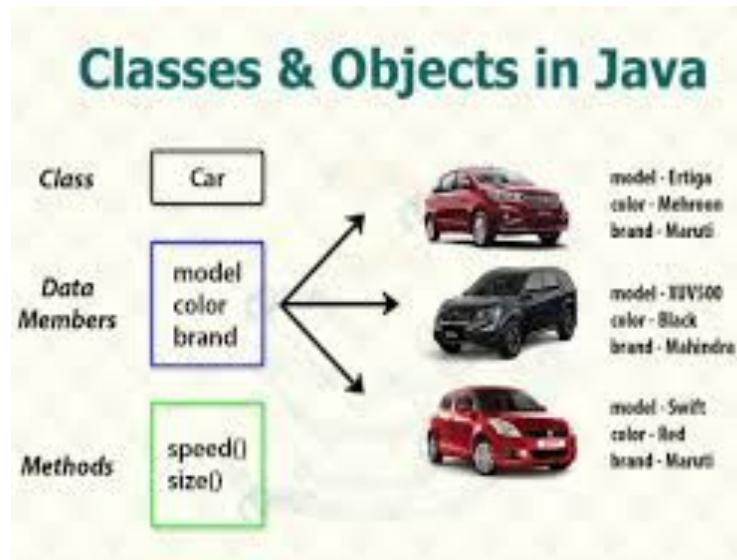
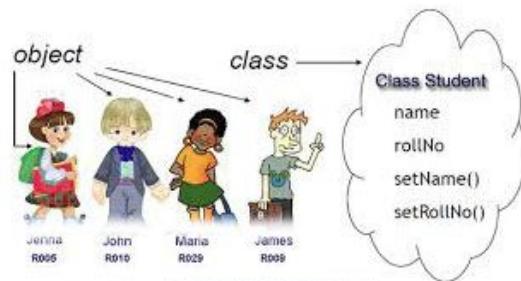
Toda Classe herda Object

A Classe serve para a definição de um Objeto

Uma classe deve ser coesa e ter baixo acoplamento



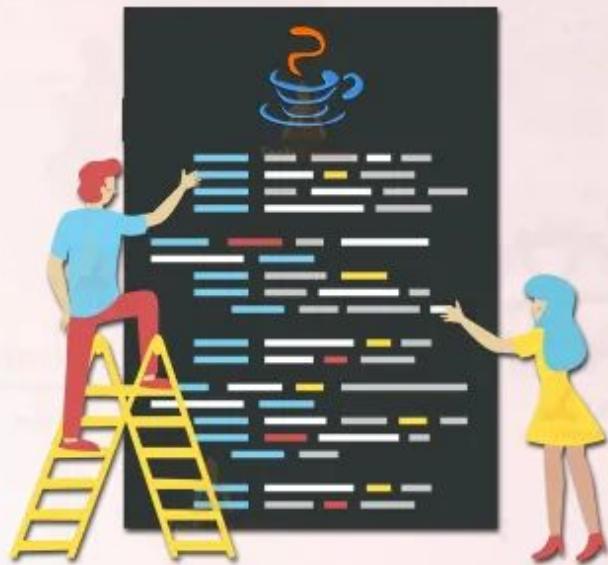
Objeto é a instância de uma classe





Como criar um Objeto

Ways to Create Object in Java



01

By new keyword

02

By newInstance() method

03

By clone() method

04

By deserialization

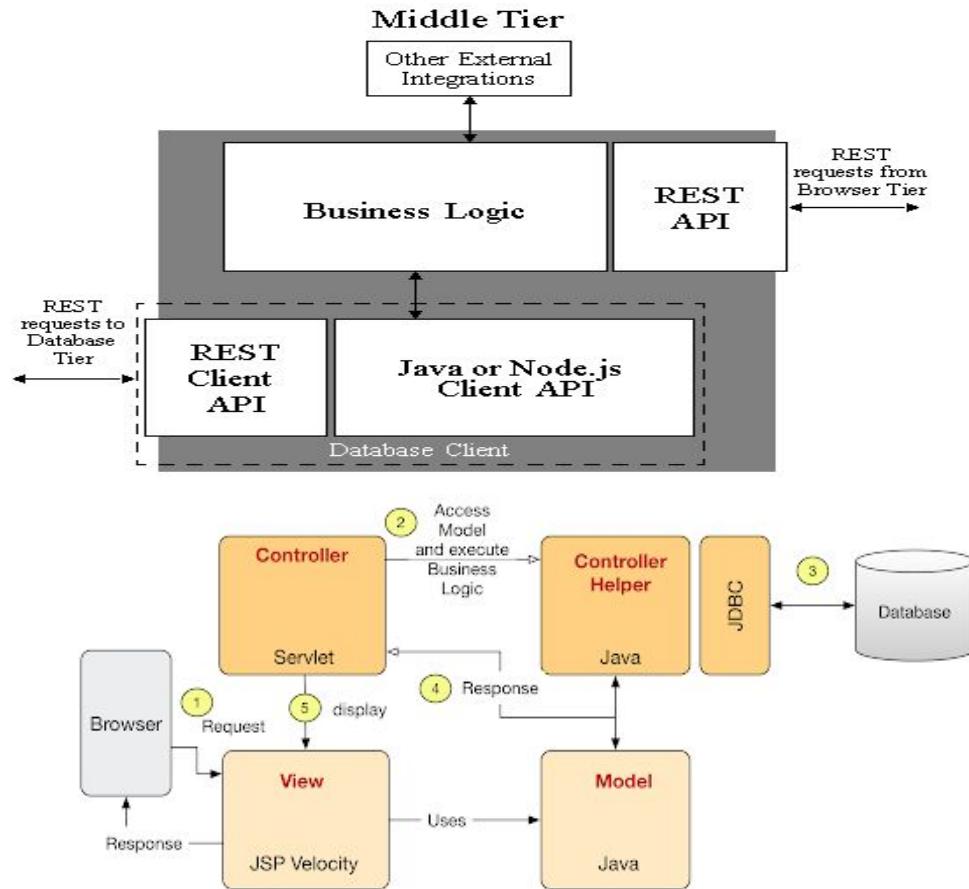
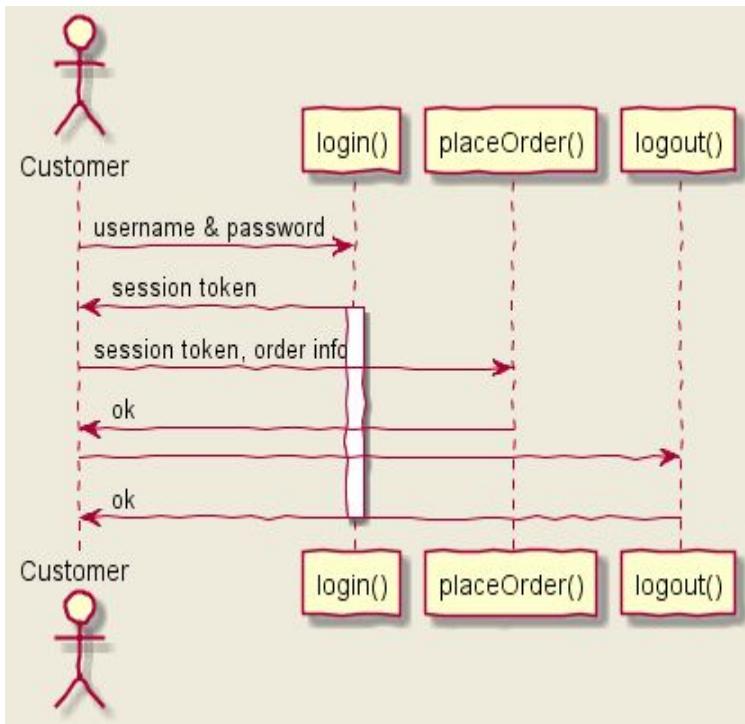
05

By factory method

01



Papel de um Objeto Java no mundo dos sistemas





Estrutura de uma Classe

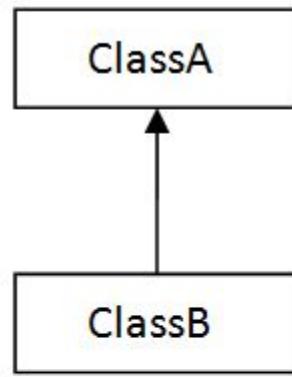
```
package com.mm;           package declaration
import java.util.Date;    import statements
5.  /**               comments
   * @author tutorialkart.com
   */
public class ProgramStructure { class name
10.    int repetitions = 3; global variable
15.    public static void main(String[] args){
        ProgramStructure programStructure = new ProgramStructure();
        programStructure.printMessage("Hello World. I started learning Java.");
      }
20.    public void printMessage(String message){
        Date date = new Date(); variable local to the method
        for(int index=0;index < repetitions;index++){
          System.out.println(message+" From "+date.toGMTString());
        }
      }
}
```

The diagram illustrates the structure of a Java class. It highlights various components with callout boxes and labels:

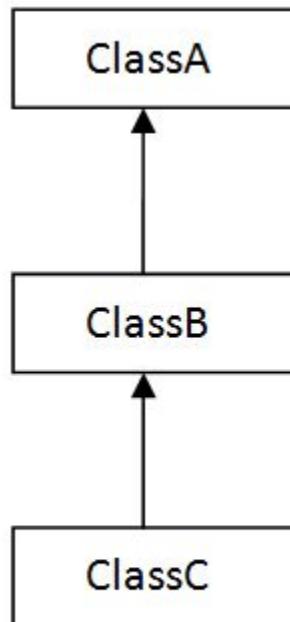
- package declaration:** Points to the first line: `package com.mm;`
- import statements:** Points to the second line: `import java.util.Date;`
- comments:** Points to the multi-line Javadoc-style comment block starting with `/**`.
- class name:** Points to the `public class` keyword followed by the class name `ProgramStructure`.
- global variable:** Points to the declaration of the `repetitions` variable.
- main method:** Points to the `main` method definition, which includes the parameter list and the body of the method.
- variable local to the method:** Points to the declaration of the `Date` variable within the `printMessage` method.
- method:** Points to the entire `printMessage` method definition.
- variable local to the for loop:** Points to the declaration of the `index` variable within the `for` loop in the `printMessage` method.



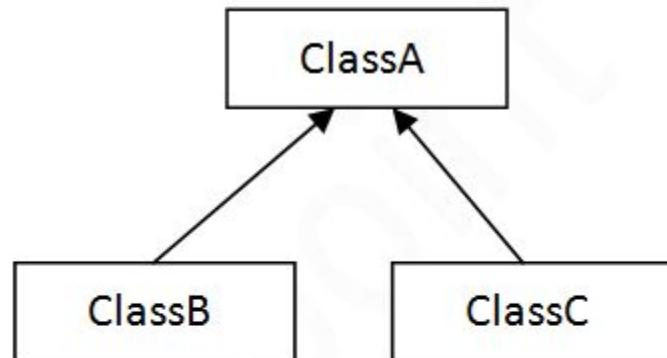
Herança (extends)



1) Single



2) Multilevel

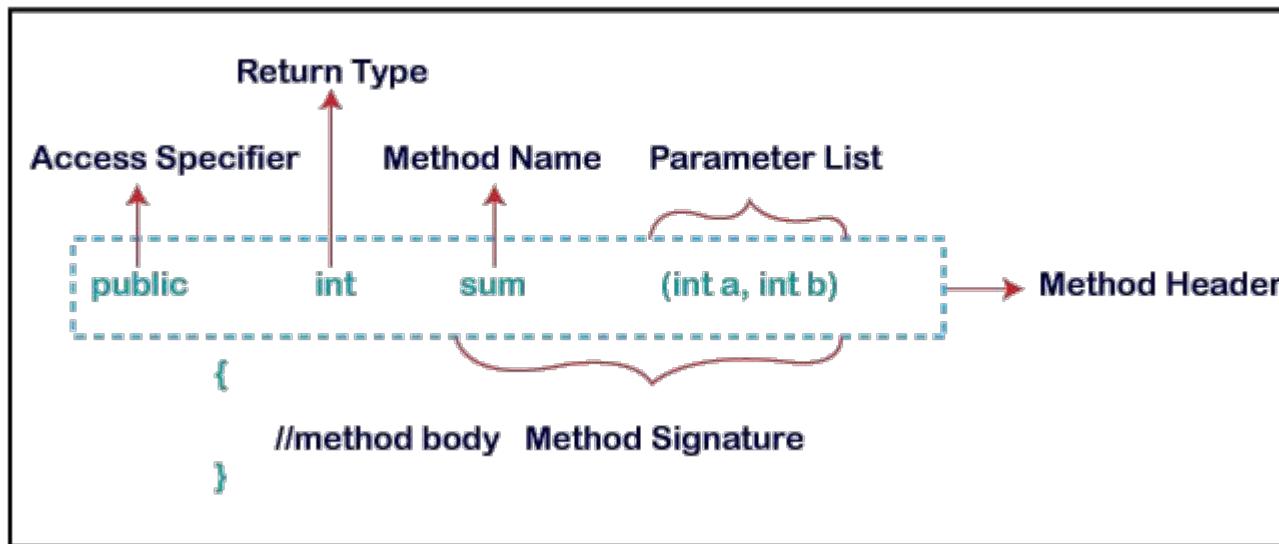


3) Hierarchical



Métodos

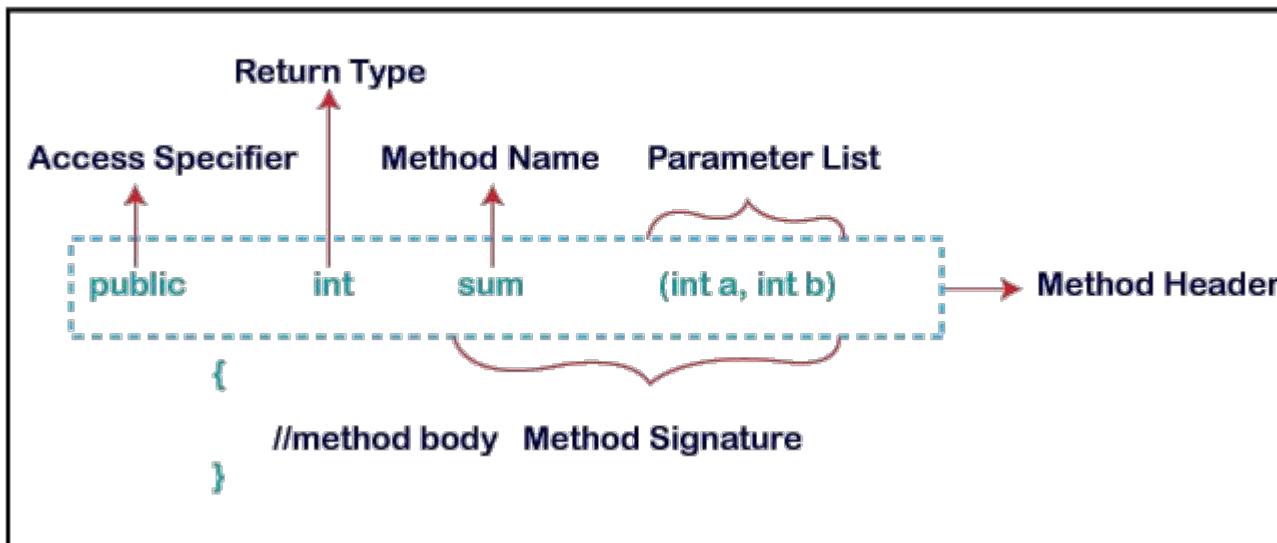
Method Declaration





Métodos

Method Declaration





Sobrescrita e Sobrecarga dos Métodos

Overriding

```
class Dog{  
    public void bark(){  
        System.out.println("woof ");  
    }  
}  
  
class Hound extends Dog{  
    public void sniff(){  
        System.out.println("sniff ");  
    }  
  
    public void bark(){  
        System.out.println("bowl");  
    }  
}
```

Same Method Name,
Same parameter

Overloading

```
class Dog{  
    public void bark(){  
        System.out.println("woof ");  
    }  
  
    //overloading method  
    public void bark(int num){  
        for(int i=0; i<num; i++)  
            System.out.println("woof ");  
    }  
}
```

Same Method Name,
Different Parameter



Atributos de Classe, Variável local e parâmetros

Attributes, local variables, parameters

```
public class Person
{
    private String name; //attribute (instance variable)

    public void method1(String yourName)//parameter
    {
        String myName; // local variable

        ... this.name; //? #1
        ... this.myName; //? #2
        ... this.yourName; //? #3

        ... name; //? #4
        ... myName; //? #5
        ... yourName; //? #6

    }
}
```



Classes, Métodos e atributos estáticos

Classe estática

Método estático

Atributos/variáveis estáticas

Bloco estáticos



Classes estáticas

```
InnerClassDemo.java

public class InnerClassDemo {
    public static void main(String[] args) {
        // Accessing the Static Inner class
        InnerClassDemo.StaticInnerClass innerClass = new StaticInnerClass();
        System.out.println("StaticInnerClass value : " + innerClass.getValue());
        // Accesing the Normal Inner class
        InnerClassDemo.NormalInnerCalss normalInnerCalss = new InnerClassDemo().new NormalInnerCalss();
        System.out.println("normalInnerCalss Value : "+ normalInnerCalss.getValue());
    }
    static class StaticInnerClass {
        int a = 10;
        public int getValue() {
            return a;
        }
    }
    class NormalInnerCalss {
        int instVar = 20;
        public int getValue() {
            return instVar;
        }
    }
}
```



Métodos estáticos

Os métodos estáticos podem ser acessados diretamente usando o nome da classe.

Os métodos estáticos não podem ser substituídos.

Os métodos não estáticos podem acessar métodos estáticos apenas usando o nome da classe.

Os métodos estáticos também podem acessar os métodos não estáticos usando a instância da classe.

Os métodos estáticos e não estáticos não são acessados diretamente.

Um método estático não pode se referir a “this” ou “super” em qualquer lugar.



Atributos/Variáveis estáticas

Variáveis estáticas são declaradas com a palavra-chave static.

Variáveis estáticas também são chamadas de variáveis de classe.

As variáveis de classe pertencem a toda a classe e não a uma instância específica da classe.

Uma única variável estática pode ser compartilhada por todas as instâncias de uma classe.

Não podemos acessar as variáveis estáticas dos métodos normais.

Variáveis de classe são alocadas na memória apenas uma vez no momento do carregamento da classe, e que pode ser comumente acessado por todas as instâncias da classe.

Variáveis estáticas são alocadas na memória do pool estático.

Como a memória para as variáveis da classe é alocada no momento do carregamento da própria classe, podemos acessar as variáveis estáticas diretamente com o próprio nome da classe.



Blocos estáticos

Temos diferentes tipos de blocos em Java, como bloco de inicialização, bloco sincronizado e bloco estático.

Cada bloco tem sua própria importância.

Aqui, um bloco estático é um bloco de instruções, que é definido usando a palavra-chave static.

```
static{
    //Code
}
```



Objetos e referência

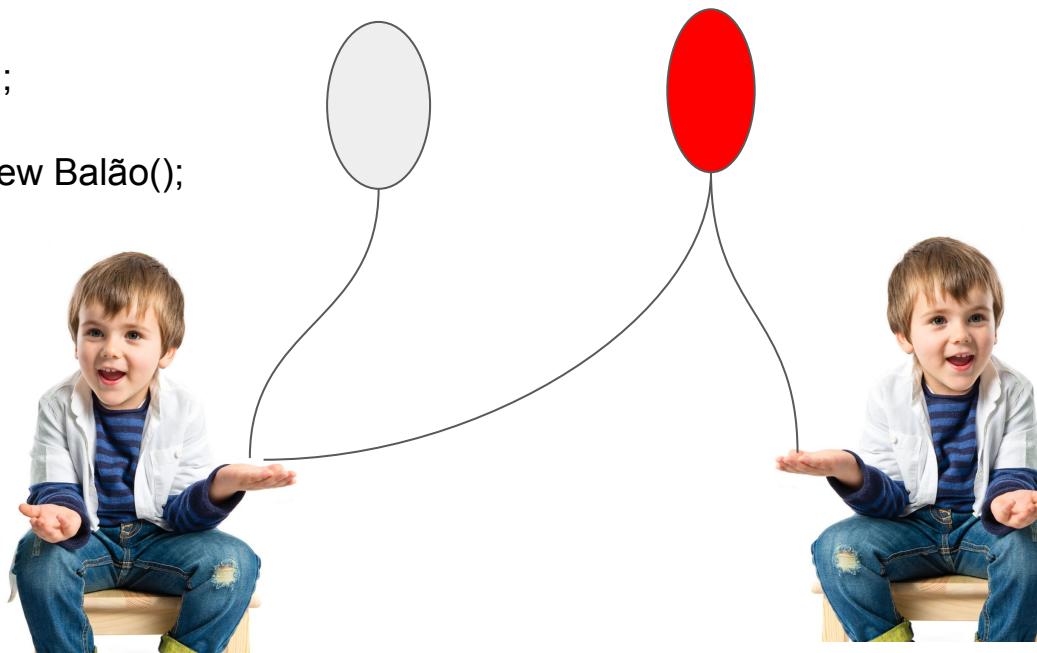
Uma referência é utilizada para armazenar o endereço de um objeto alocado na memória.
Pela referência é manipulado o estado do objeto.
Pode ser um atributo, variável local ou argumentos de métodos.

Balão criança = null;

Balão criança = new Balão();

criança = new Balão();

Balão criança2 = criança;

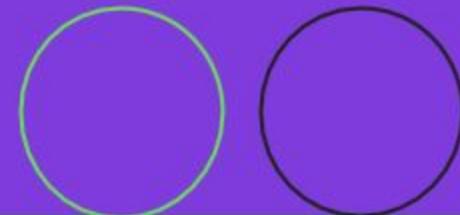




Encerramos por hoje !!!



POO



Continuando o entendimento



Construtores

um bloco de código que executa na instanciação de uma classe

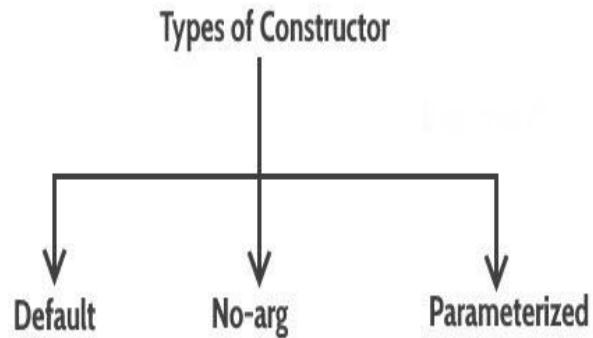
Eles são invocados implicitamente

Toda Classe já possui um construtor default herdado do Class

Construtores não possuem retornos

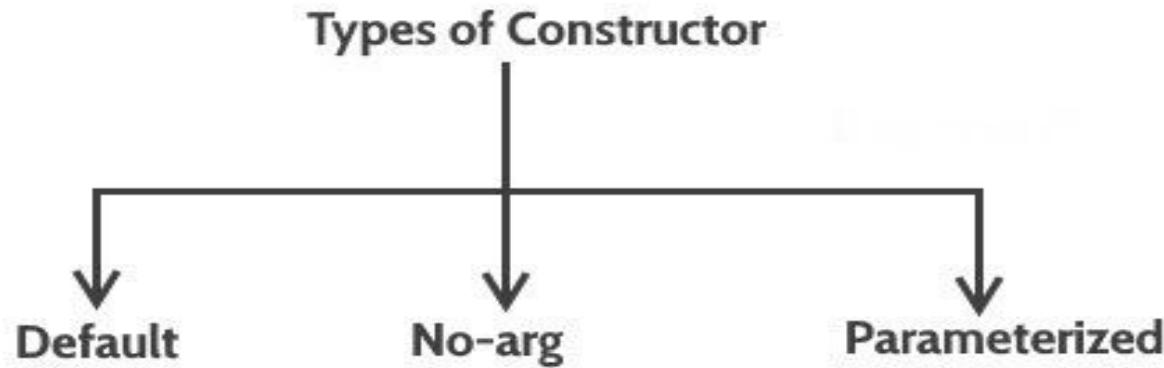
Devem conter o mesmo nome da Classe

Podem conter inúmeros parâmetros respeitando as regras da sobrecarga.





Construtores



Constructor



Blueprint (Class)

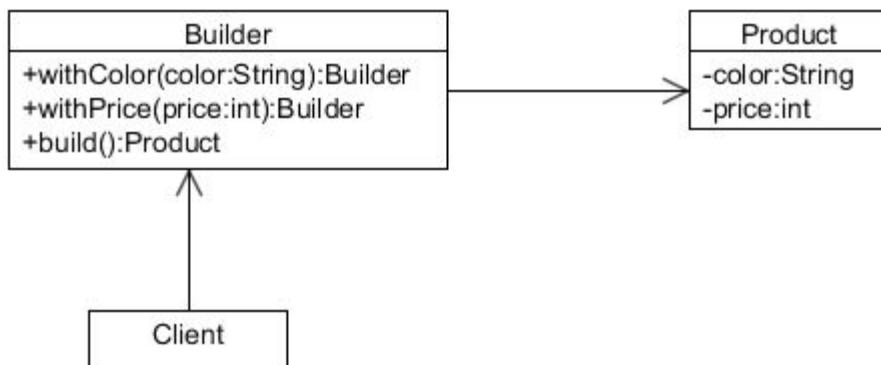
House (Object)



Padrão Builder

Responsabilidade de criar o objeto

Remove a complexidade de quando o objeto necessita de vários atributos para ser criado





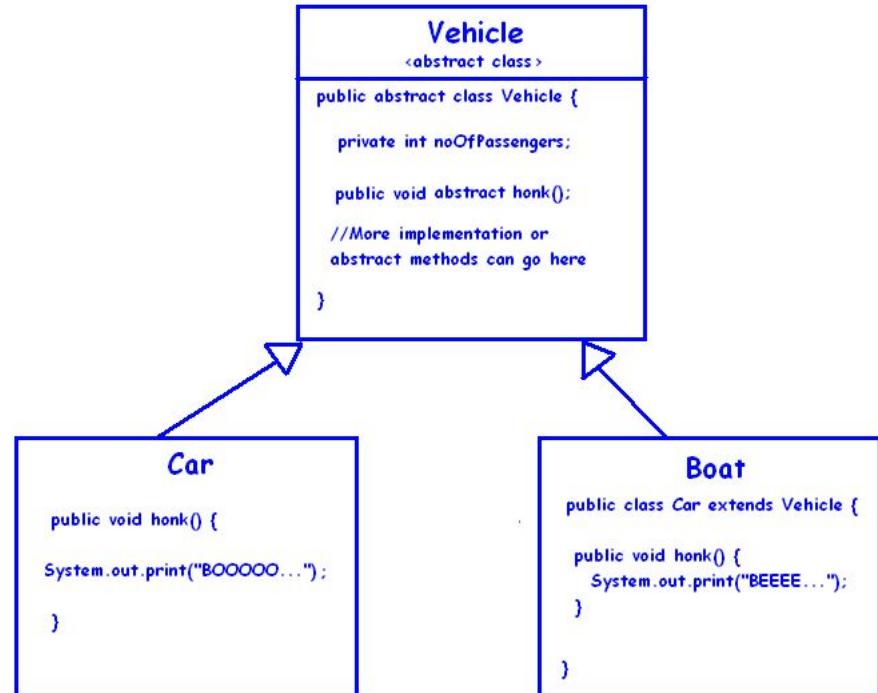
Classes e métodos Abstratas

Classes Abstratas servem como contrato

Não se pode instanciar diretamente uma classe abstrata

Pode conter implementação

Métodos abstratos não tem implementação





Classes, métodos, atributos/variáveis e final

Classes final não pode ser herdada

Métodos final não podem ser sobrescritos

Atributos/Variáveis final são constantes que não podem mudar o valor



Pacotes e visibilidade

Visibilidade	public	protected	default	private
A partir da mesma classe	✓	✓	✓	✓
Qualquer classe no mesmo pacote	✓	✓	✓	✗
Qualquer classe filha no mesmo pacote	✓	✓	✓	✗
Qualquer classe filha em pacote diferente	✓	✓	✗	✗
Qualquer classe em pacote diferente	✓	✗	✗	✗



Encapsulamento e a proteção dos atributos

Encapsulation in Java

- A way to achieve abstraction for objects' data.
- Hides object properties from outer world.
- Provides methods to get/set object data.
- Also called "data-hiding".
- Advantages:
 - Loosely coupled code
 - Better access control and security
 - Reusable code
 - Easy to test





Polimorfismo

polimorfismo é a capacidade de um objeto ser referenciado de diversas formas diferentes e com isso realizar as mesmas tarefas de diferentes formas.

Polimorfismo significa "muitas formas", é o termo definido em linguagens orientadas a objeto, como por exemplo Java, C# e C++, que permite ao desenvolvedor usar o mesmo elemento de formas diferentes.

Polimorfismo denota uma situação na qual um objeto pode se comportar de maneiras diferentes ao receber uma mensagem. No Polimorfismo temos dois tipos:

- Polimorfismo Estático ou Sobrecarga
- Polimorfismo Dinâmico ou Sobreposição



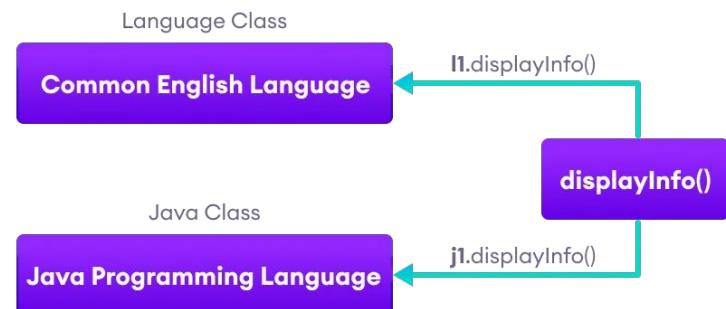
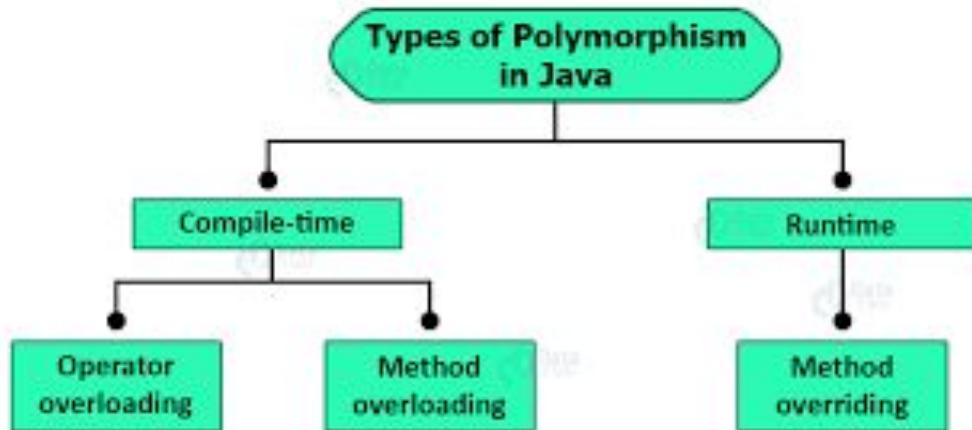
Polimorfismo

Boa parte dos padrões de projeto de software baseia-se no uso de polimorfismo, por exemplo: Abstract Factory, Composite, Observer, Strategy, Template Method, etc.

É notável a importância do Polimorfismo para a redução de código, simplicidade, flexibilidade, etc. O polimorfismo é utilizado em diversas refatorações e muitos Padrões de Projetos, portanto entendê-lo é fundamental para qualquer desenvolvedor.



Polimorfismo





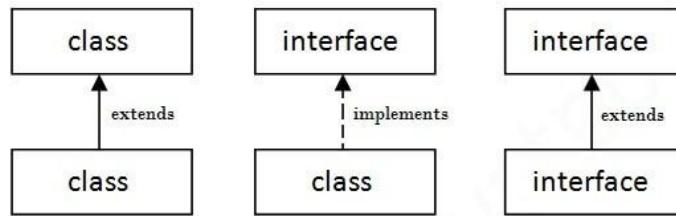
Interfaces

O papel da interface é criar um contrato

Não se pode criar objetos com interfaces

Uma interface pode herdar uma ou várias interfaces

Uma Classe pode implementar uma ou várias interfaces





var-args

Recurso implantado na versão Java 5

Permite a inclusão de 0 ou mais argumentos do mesmo tipo



Genéricos

Introduzido no java 5

Criado para evitar erros em runtime

Necessidade de cast na recuperação dos dados com tipagem vinculada na declaração

Wildcards comuns *List<?>*, *List<? extends Number>* e *List<? super Integer>*



Enumeration

Introduzido no java 5

O tipo **Enum** no Java, é um tipo de valores constantes, pré definidas, que servem para várias situações do dia a dia de um programador.

A sintaxe de criação de um **Enum** no Java é muito semelhante a criação de uma classe, porém no lugar de usar class na declaração, usamos enum.

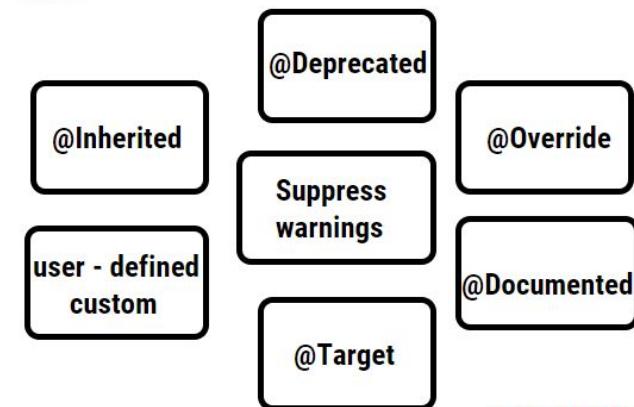


Anotações

Java Annotations



Predefined /standard annotations





Anotações

As anotações ajudam a definir metadados no código de maneira padronizada. Além disso, as anotações ajudam a fornecer instruções para o seu compilador java seguir ao compilar esse código java.

Pontos importantes a serem lembrados são que todas as anotações estendem a interface `java.lang.annotation.Annotation`. Além disso, as anotações não podem incluir nenhuma cláusula `extended`.



Anotações

Ao usar as anotações, usamos o sinal '@' seguido pelo nome de sua anotação para que o compilador a trate como uma anotação. É importante notar que as anotações podem ser adicionadas antes de uma declaração de -

- Classe
- Variável membro
- construtor
- método
- parâmetros de métodos/construtores
- Variável local.



Conceito SOLID

1. **S**ingle Responsibility
2. **O**pen/Closed
3. **L**iskov Substitution
4. **I**nterface Segregation
5. **D**ependency Inversion



Single Responsibility

```
public class Book {  
  
    private String name;  
    private String author;  
    private String text;  
  
    //constructor, getters and setters  
  
    // methods that directly relate to the book  
    properties  
    public String replaceWordInText(String word){  
        return text.replaceAll(word, text);  
    }  
  
    public boolean isWordInText(String word){  
        return text.contains(word);  
    }  
}
```

```
public class Book {  
    //...  
  
    void printTextToConsole(){  
        // our code for formatting and  
        printing the text  
    }  
}
```

```
public class BookPrinter {  
  
    // methods for outputting text  
    void printTextToConsole(String text){  
        //our code for formatting and printing the  
        text  
    }  
  
    void printTextToAnotherMedium(String  
text){  
        // code for writing to any other location..  
    }  
}
```



Open/Closed

```
public class BookPrinter {  
  
    // methods for outputting text  
    void printTextToConsole(String  
text){  
        //our code for formatting and  
printing the text  
    }  
  
    void  
printTextToAnotherMedium(String  
text){  
        // code for writing to any other  
location..  
    }  
}
```

```
public class  
SuperCoolGuitarWithFlames  
extends Guitar {  
  
    private String flameColor;  
  
    //constructor, getters + setters  
}
```



Liskov Substitution

```
public interface Car {  
  
    void turnOnEngine();  
    void accelerate();  
}
```

```
public class MotorCar implements Car {  
  
    private Engine engine;  
  
    //Constructors, getters + setters  
  
    public void turnOnEngine() {  
        //turn on the engine!  
        engine.on();  
    }  
  
    public void accelerate() {  
        //move forward!  
        engine.powerOn(1000);  
    }  
}
```

```
public class ElectricCar implements Car {  
  
    public void turnOnEngine() {  
        throw new  
            AssertionError("I don't have an engine!");  
    }  
  
    public void accelerate() {  
        //this acceleration is crazy!  
    }  
}
```



Interface Segregation

```
public interface BearKeeper {  
    void washTheBear();  
    void feedTheBear();  
    void petTheBear();  
}
```

```
public interface BearCleaner {  
    void washTheBear();  
}  
  
public interface BearFeeder {  
    void feedTheBear();  
}  
  
public interface BearPetter {  
    void petTheBear();  
}
```

```
public class BearCarer implements BearCleaner, BearFeeder {  
  
    public void washTheBear() {  
        //I think we missed a spot...  
    }  
  
    public void feedTheBear() {  
        //Tuna Tuesdays...  
    }  
}
```



Dependency Inversion

```
public class Windows98Machine {  
  
    private final StandardKeyboard keyboard;  
    private final Monitor monitor;  
  
    public Windows98Machine() {  
        monitor = new Monitor();  
        keyboard = new StandardKeyboard();  
    }  
  
}
```

```
public interface Keyboard { }
```

```
public class Windows98Machine{  
  
    private final Keyboard keyboard;  
    private final Monitor monitor;  
  
    public Windows98Machine(Keyboard  
                           keyboard, Monitor monitor) {  
        this.keyboard = keyboard;  
        this.monitor = monitor;  
    }  
}
```

```
public class StandardKeyboard implements Keyboard { }
```

Recursos

APIs e controles de fluxos



Datas com java

java.util.Date

java.sql.Date

java.util.Calendar e java.util.GregorianCalendar



Formatação de Datas com java

```
SimpleDateFormat formato = new SimpleDateFormat("dd/MM/yyyy");
Date data = formato.parse("23/11/2015");
```

<https://www.devmedia.com.br/utilizando-recursos-do-java-para-formatacao-de-datas/5720>

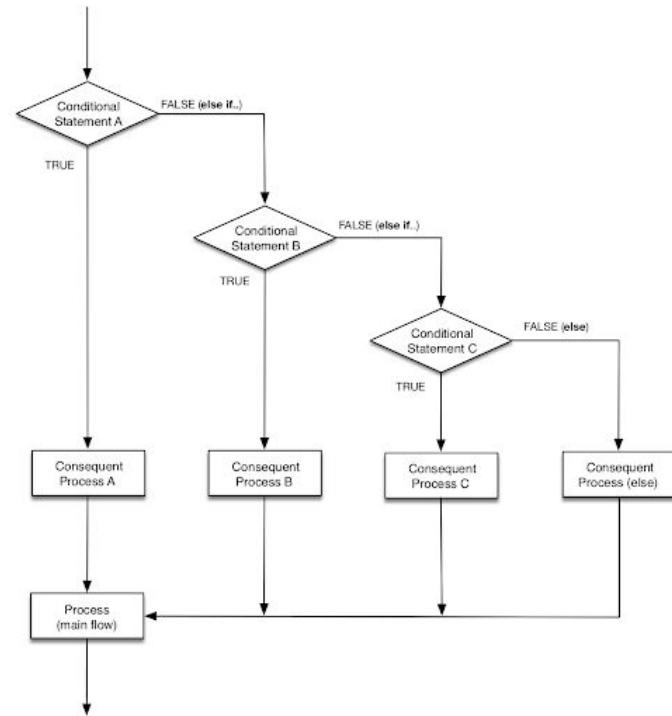
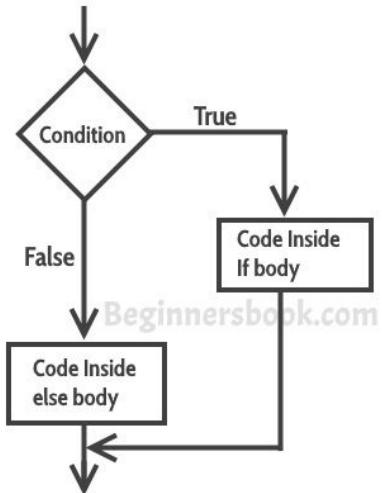
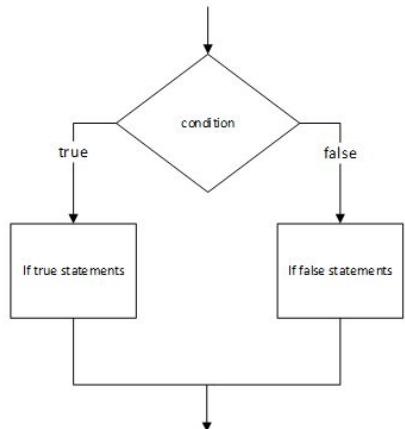


Nova biblioteca de datas

LocalDate, LocalTime e LocalDateTime

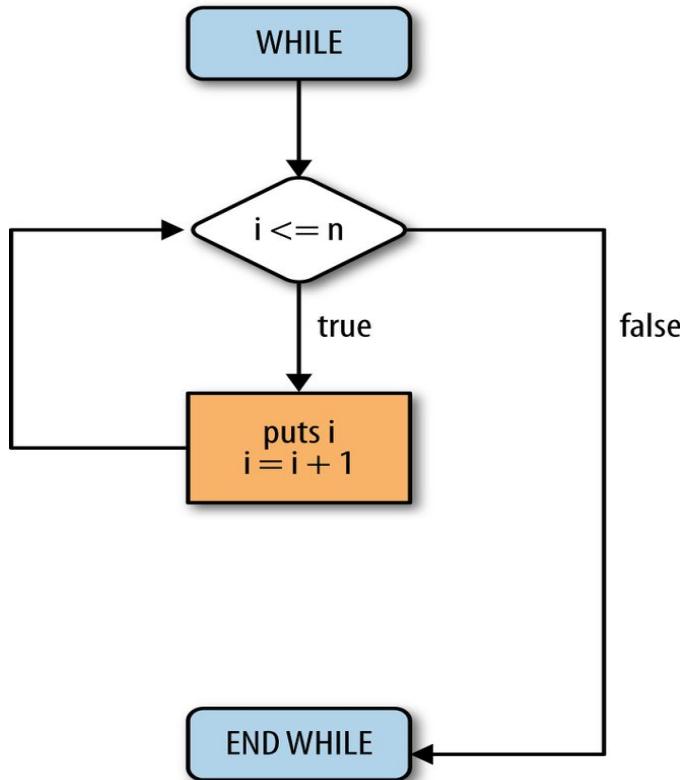


Fluxo de decisão (if, else, switch)





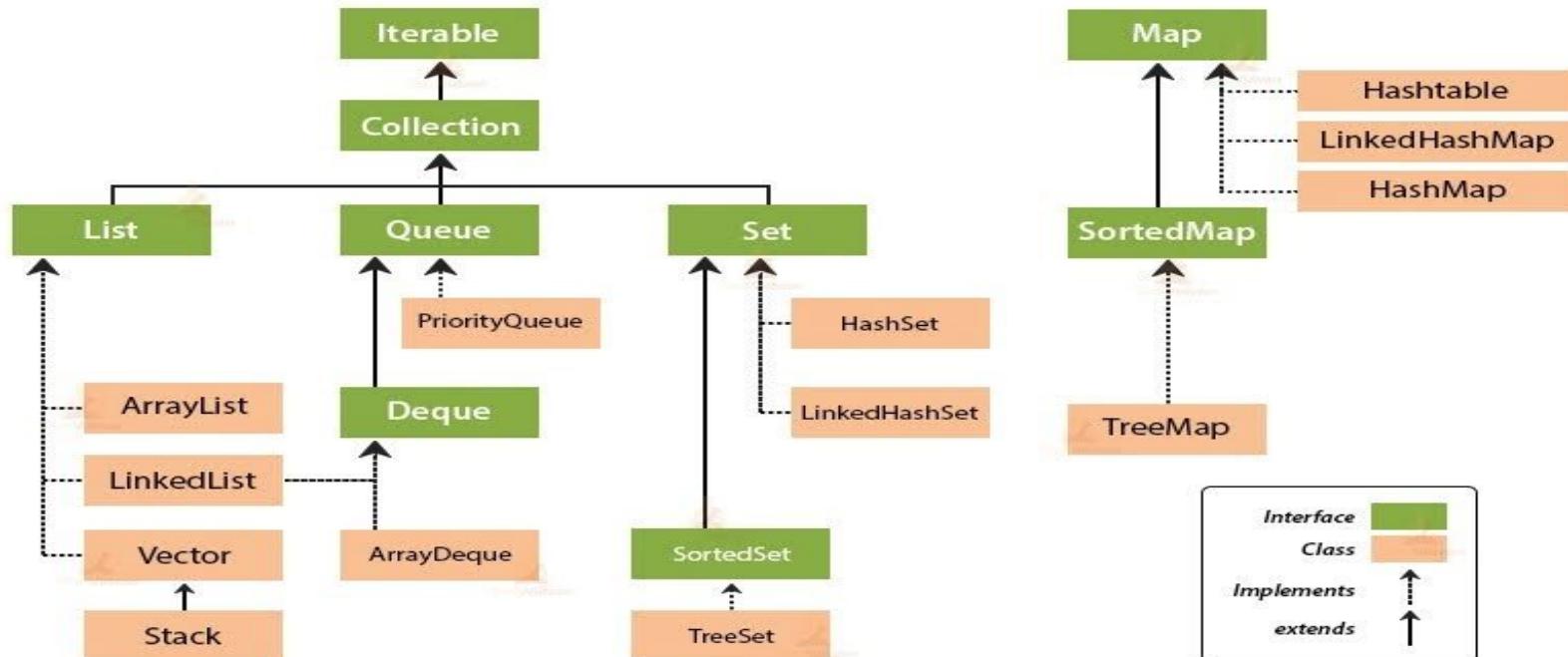
Fluxo de repetição (for/do/while/break/continue)





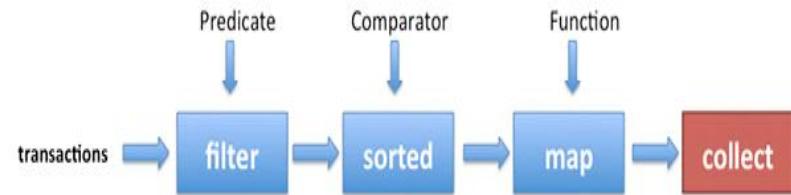
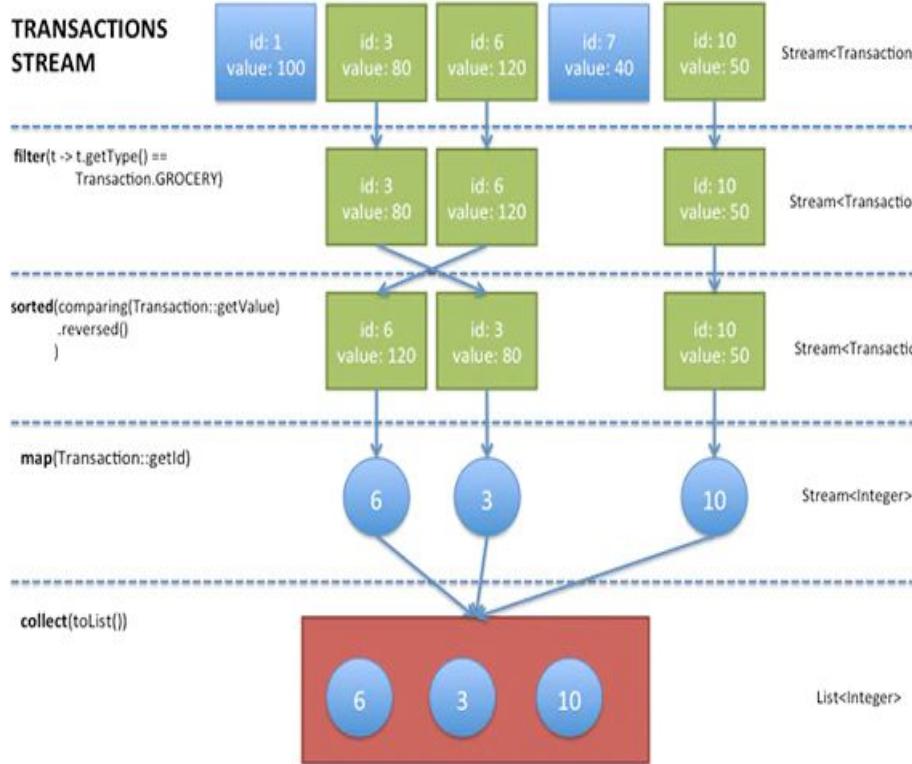
Collections

Collection Framework Hierarchy in Java

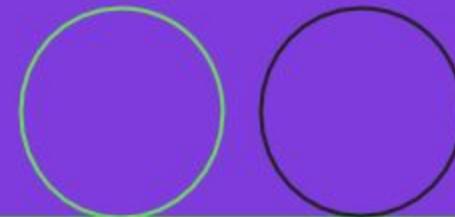




Streams



Exceções



Tratamento de exceções em Java



Tratamento de Exceções

Exceções são situações que não são esperadas ou podem ser provocadas pelo código

Existem exceções checadas e não checadas

Para capturar uma exceção deve-se proteger o código com a instrução try e/ou (catch/finally)



Tratamento de Exceções

The screenshot shows a Java code editor window titled "ReturnValueExample.java". The code demonstrates exception handling with a static method that returns an integer. A tooltip is displayed over the line "public static int returnValueFromMethod() {", indicating that the method must return an integer type. The tooltip contains two quick fix options: "Add return statement" and "Change return type to 'void'".

```
1 package in.bench.resources.exception.handling;
2
3 public class ReturnValueExample {
4
5     public static void main(String[] args) {
6
7         // invoking static method
8         returnValueFromMethod();
9     }
10
11    public static int returnValueFromMethod() {
12
13        int result = 0;
14
15        try {
16            result = 18/0;
17        }
18        catch(ArithmaticException aex){
19            System.out.println(aex.toString());
20        }
21        finally {
22            System.out.println("finally block always executed for resource clean-up");
23        }
24    }
25 }
```



Tratamento de Exceções

1. try
2. catch
3. throw
4. throws
5. finally

```
class ThrowDemo
{
    public static void main(String args[])
    {
        try
        {
            throw new ArithmeticException();
        }
        catch(ArithmeticException e)
        {
            System.out.print("Arithmetic Exception caught");
        }
    }
}
```

```
class SomeClass {
    public void method(int i) throws IllegalArgumentException {
        if (i < 0)
            throw new IllegalArgumentException();
        // ...
    }
}
```

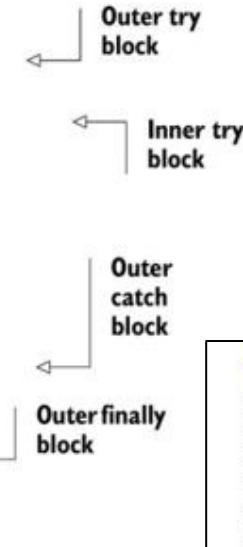
Says "this method throws the checked exception `IllegalArgumentException`".

Throws an `IllegalArgumentException`.



Tratamento de Exceções

```
import java.io.*;
public class NestedTryCatch {
    FileInputStream players, coach;
    public void myMethod() {
        try {
            players = new FileInputStream("players.txt");
            try {
                coach = new FileInputStream("coach.txt");
                //.. rest of the code
            } catch (FileNotFoundException e) {
                System.out.println("coach.txt not found");
            }
            //.. rest of the code
        }
        catch (FileNotFoundException fnfe) {
            System.out.println("players.txt not found");
        }
        finally {
            try {
                players.close();
                coach.close();
            } catch (IOException ioe) {
                System.out.println(ioe);
            }
        }
    }
}
```

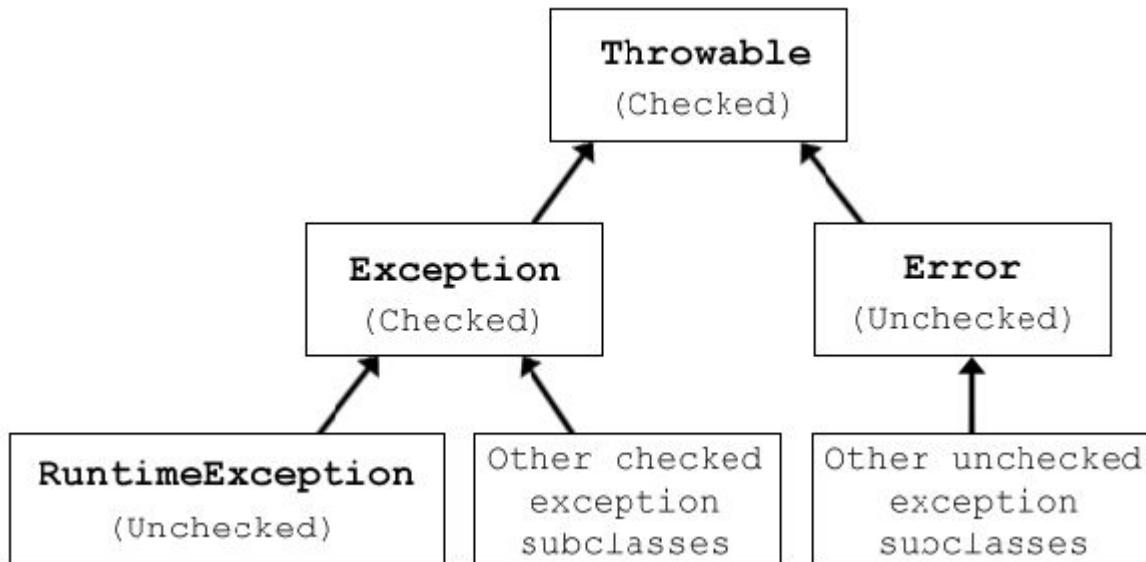


Try closeable

```
try (IncomingTrackHandler ith = f.getInstance()) {
    if (!ith.exists(iet)) {
        ith.create(iet);
        // commit to API
        ith.commit();
    }
} catch (Exception e) {
    //ith.rollback();
}
```



Tratamento de Exceções



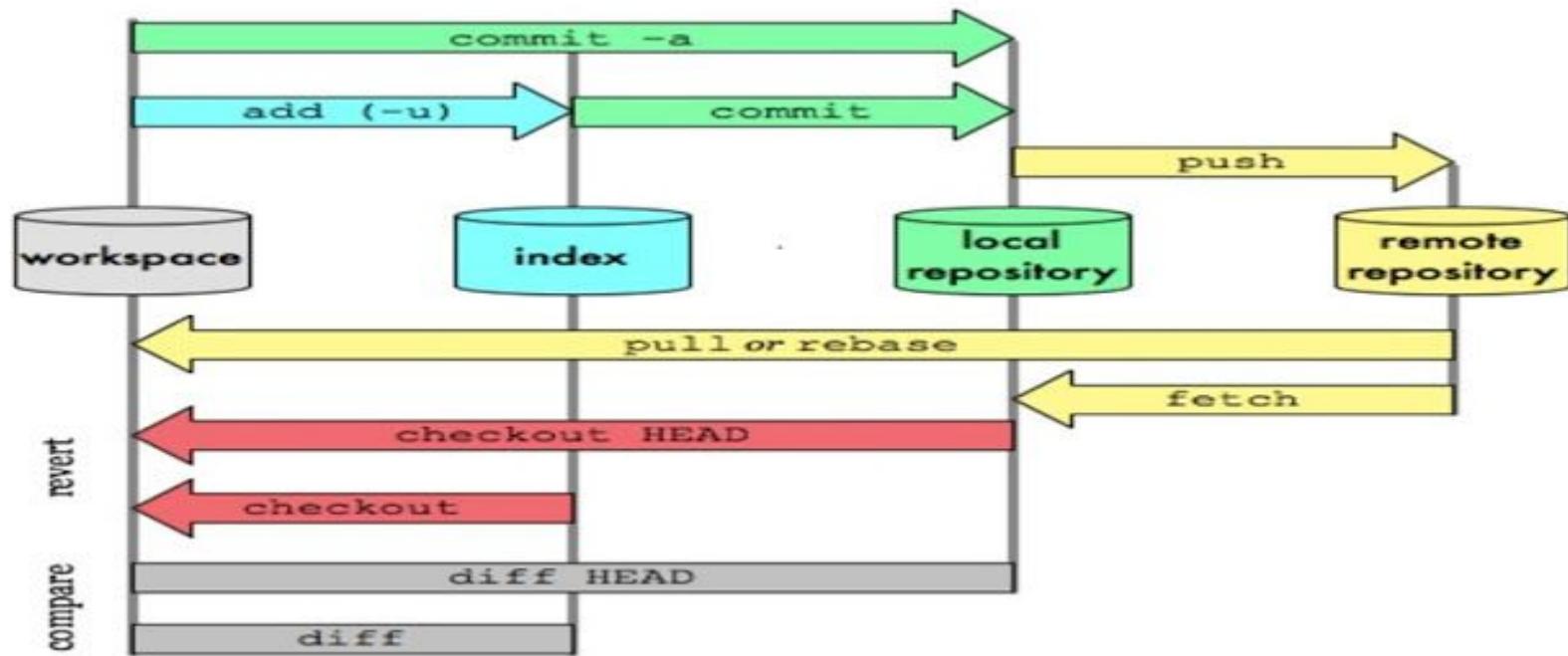
Git Flow

Gerenciamento de Branchs



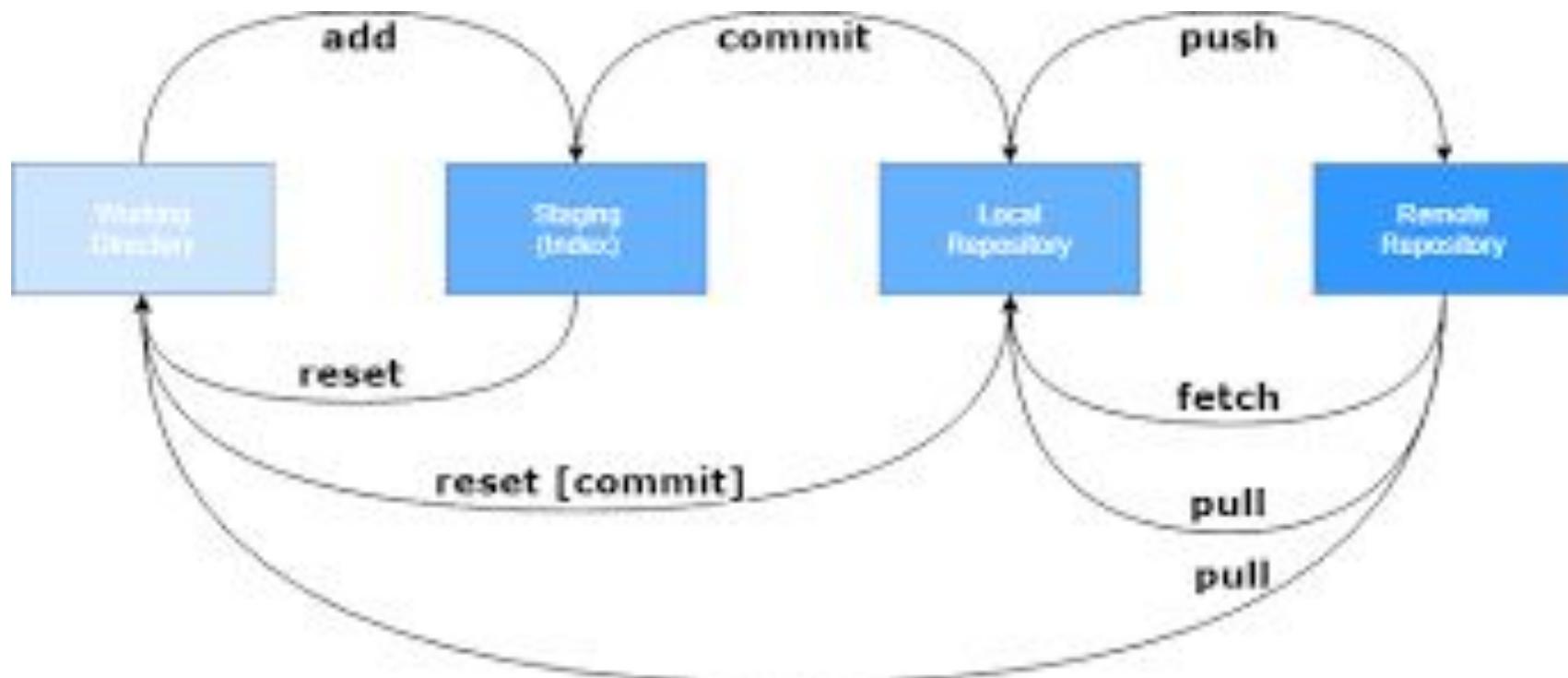
Gerenciamento de Branch com Git Flow

Git Data Transport Commands



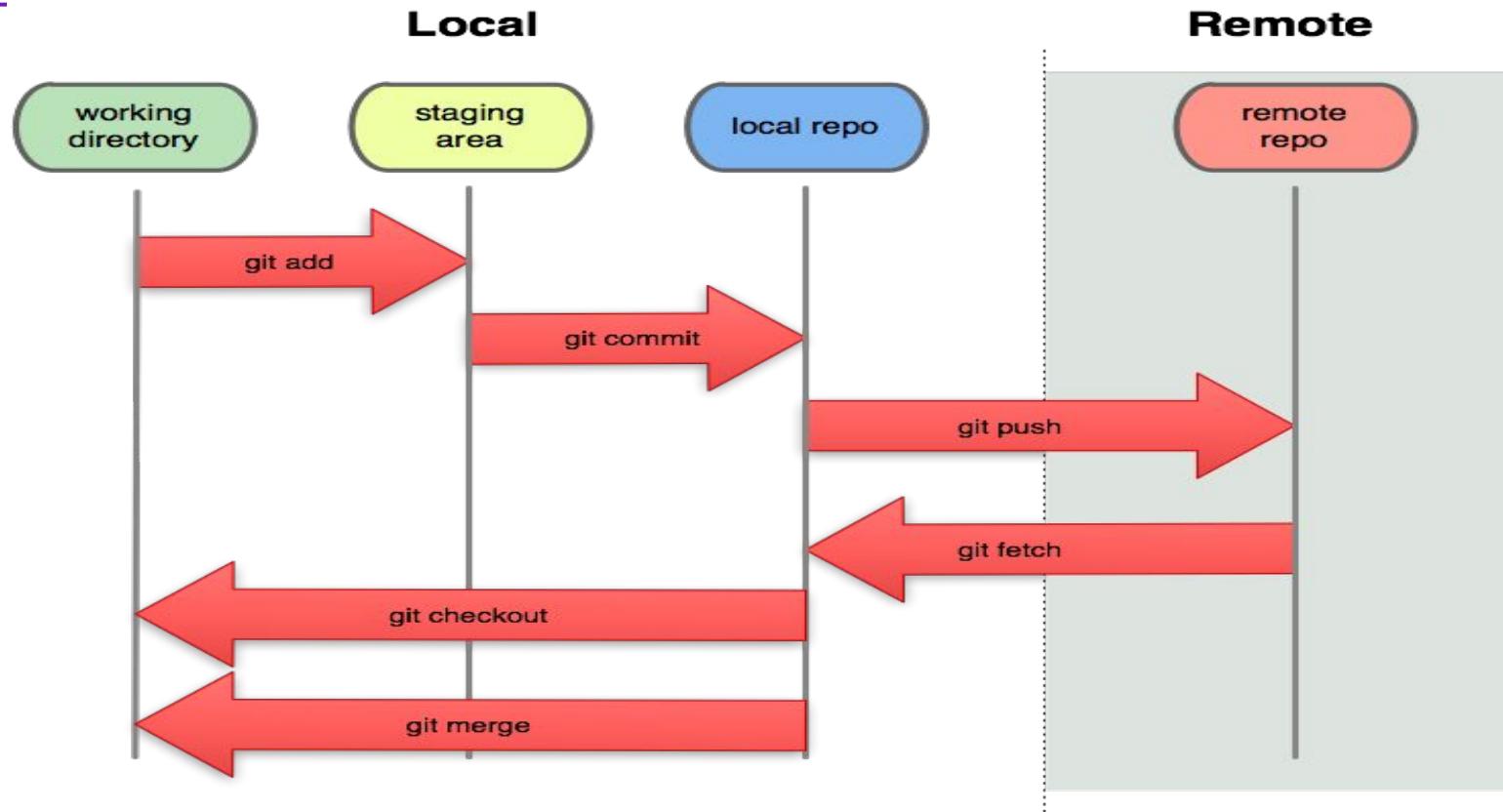


Gerenciamento de Branch com Git Flow



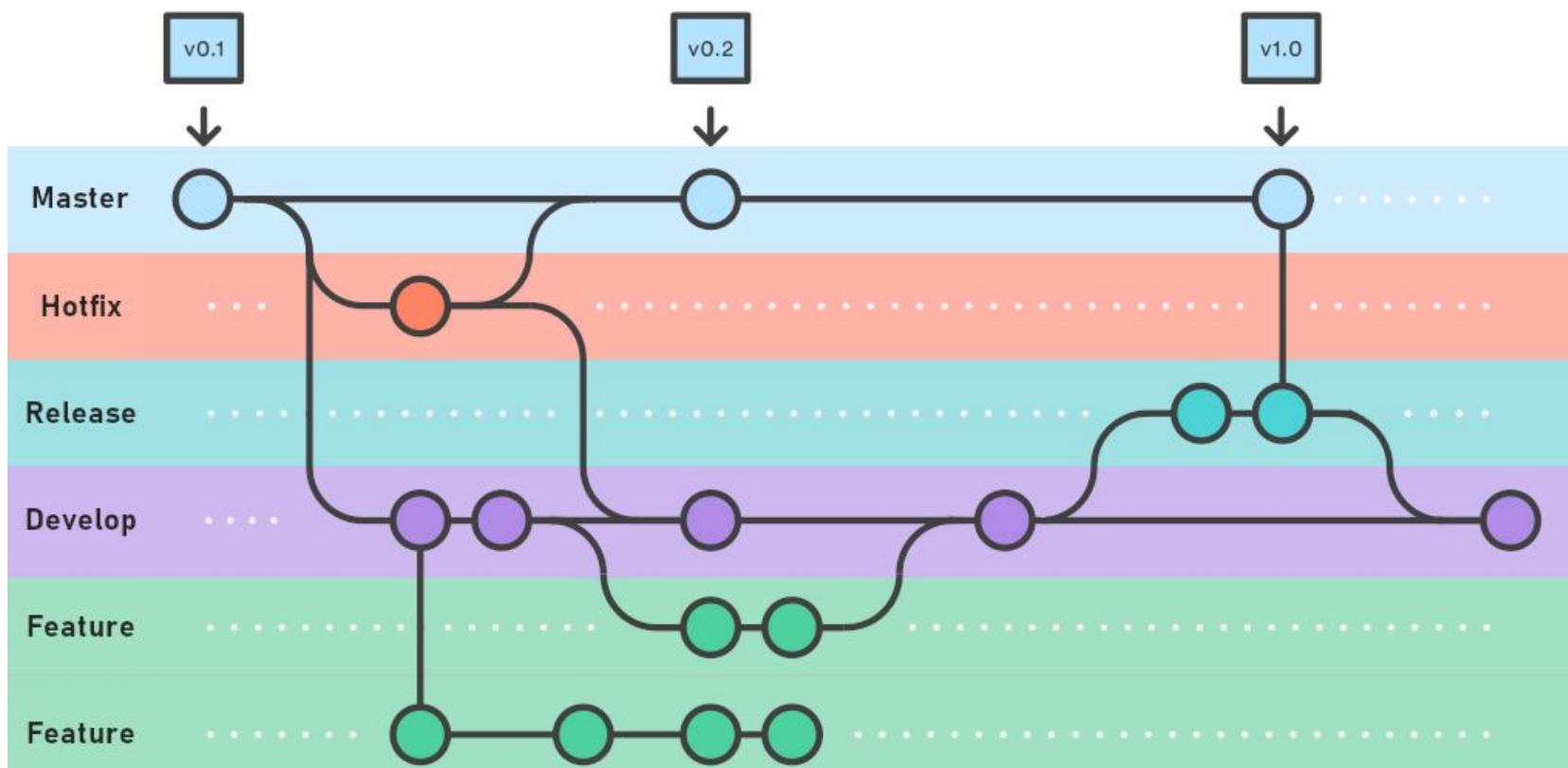


Gerenciamento de Branch com Git Flow



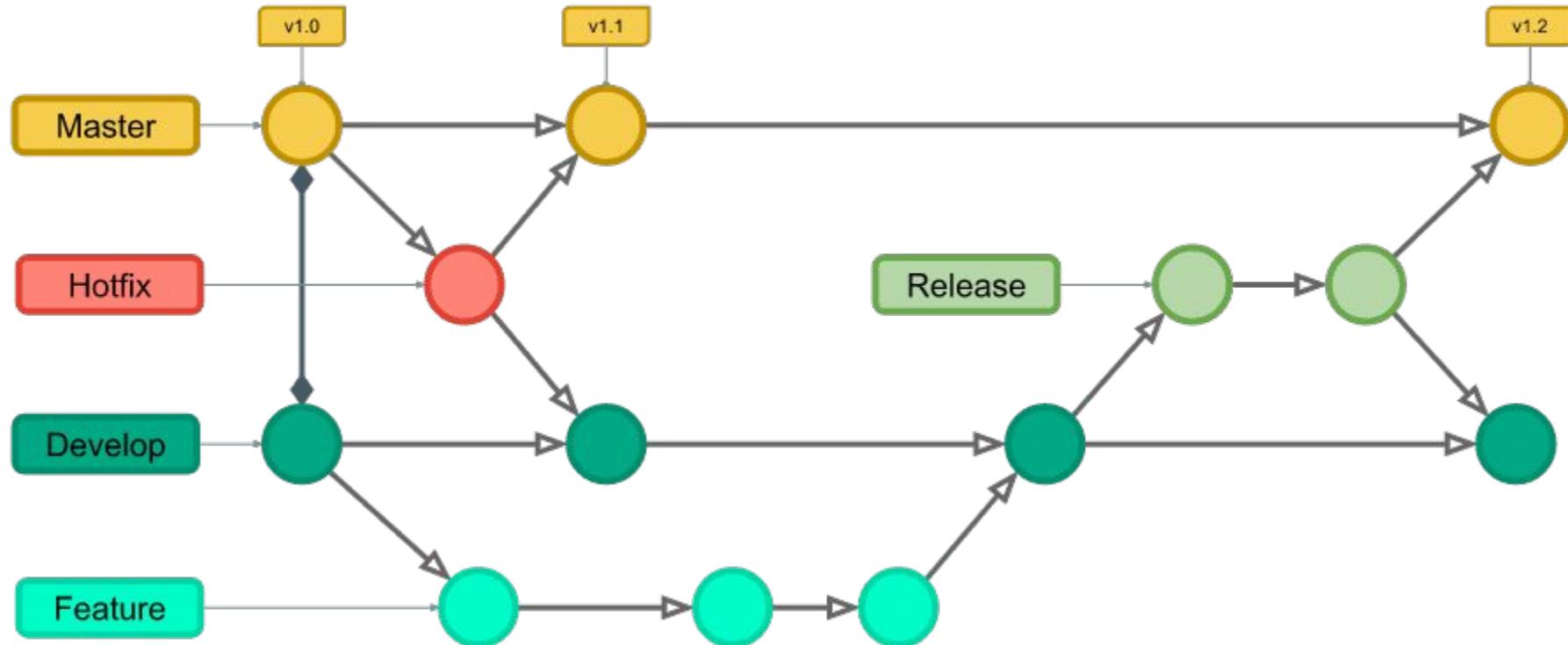


Gerenciamento de Branch com Git Flow



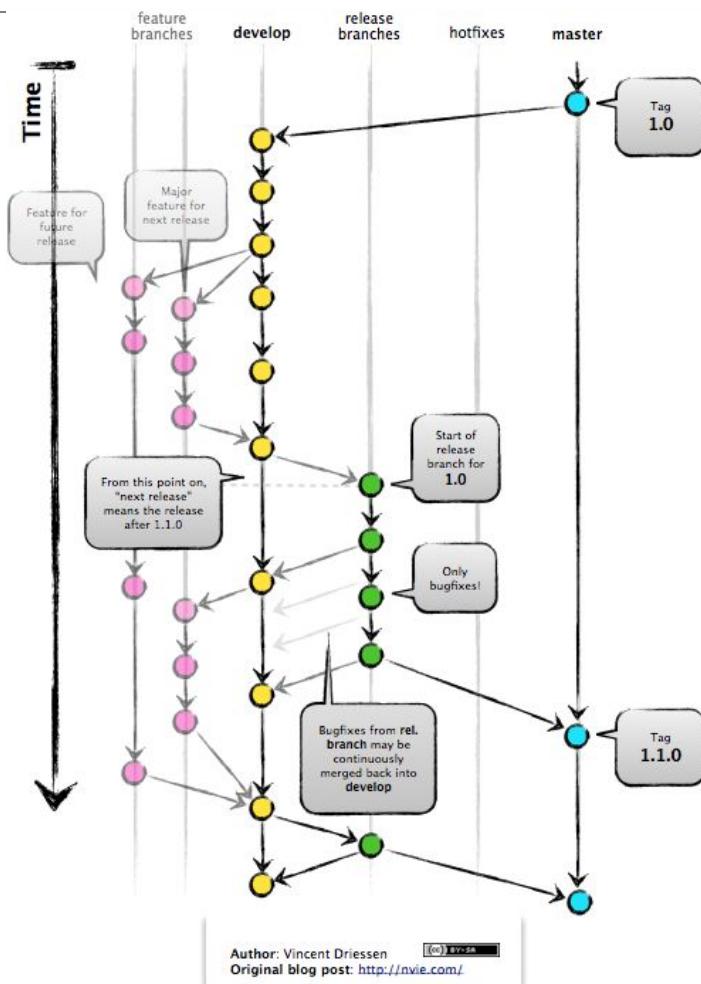


Gerenciamento de Branch com Git Flow





Gerenciamento de Branch com Git Flow





Encerramos por hoje !!!

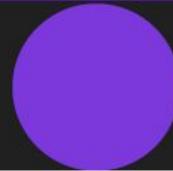




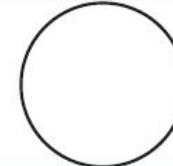
Principais Frameworks

JavaEE Vs. Spring

JavaEE



Plataforma Robusta





Servidores JavaEE

CAUCHO



IBM.

TmaxSoft
TmaxSoft Co., Ltd.

SAP



ORACLE
FUSION MIDDLEWARE
WEBLOGIC SERVER

Cosminexus

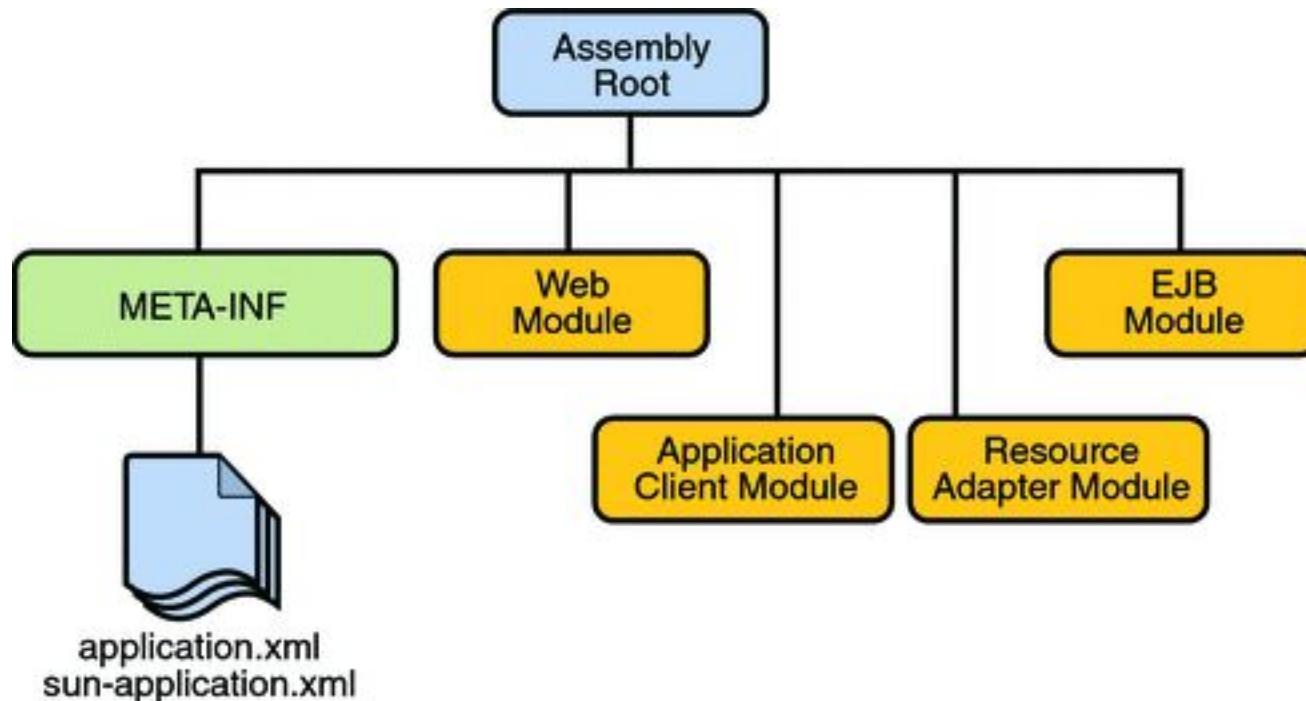
FUJITSU

APACHE
GERONIMO

OW2
Consortium

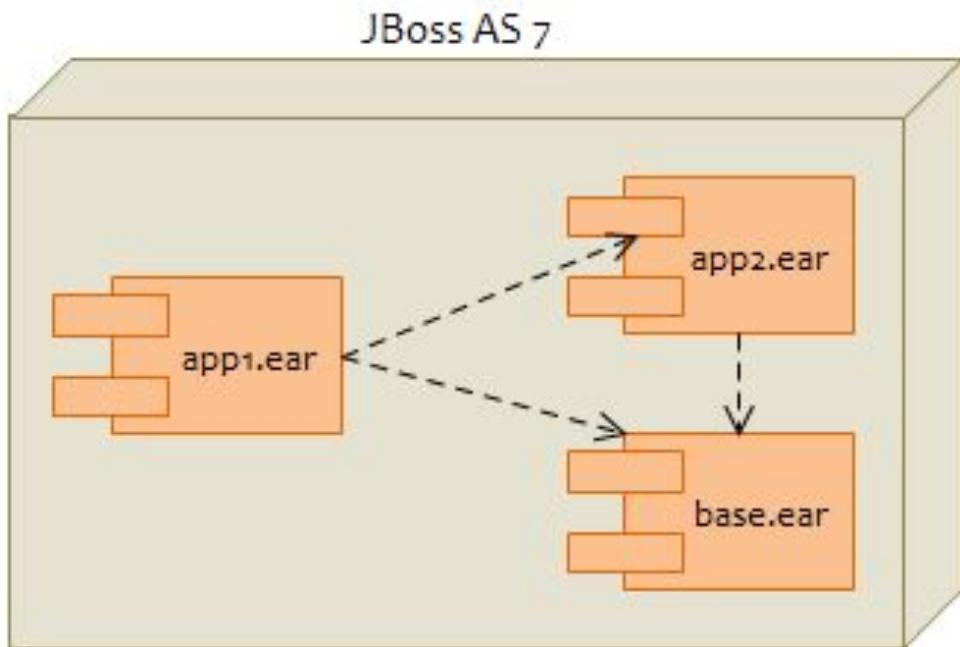


EAR package



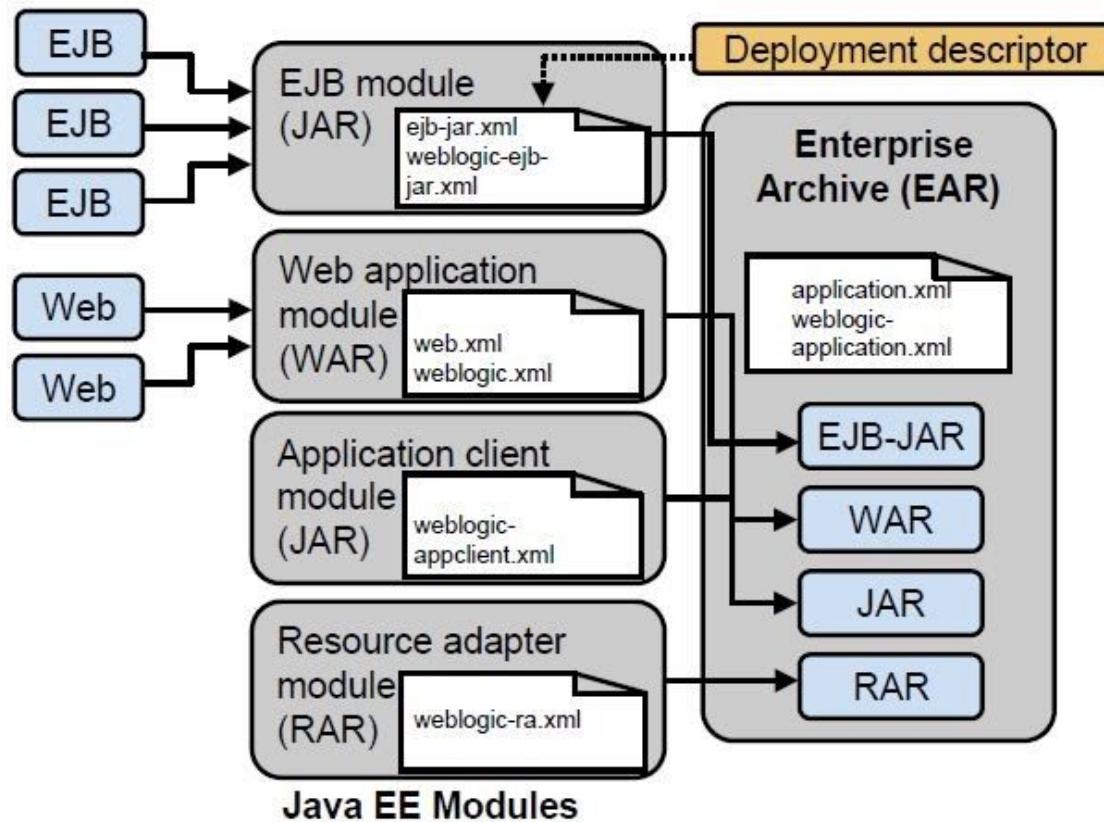


EAR package



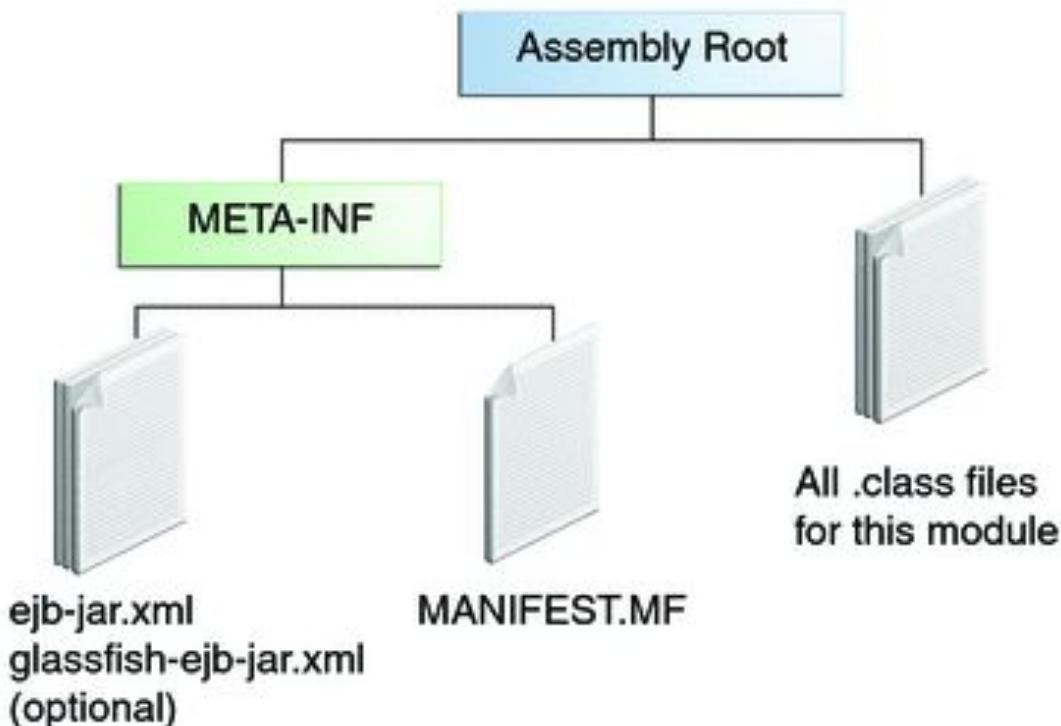


EAR package



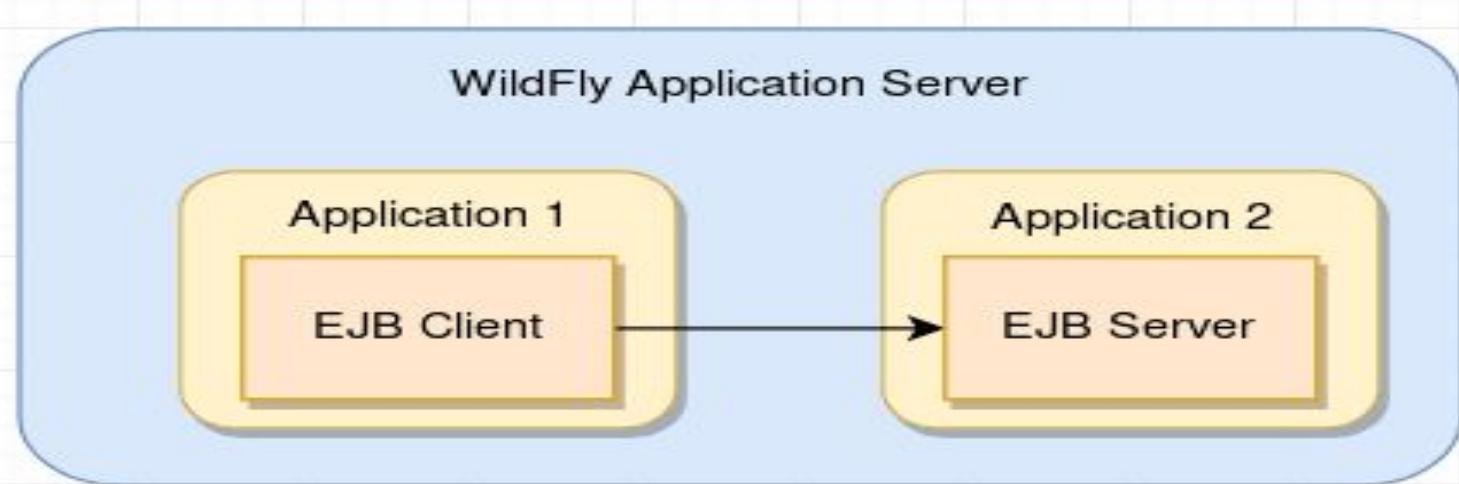


EJB





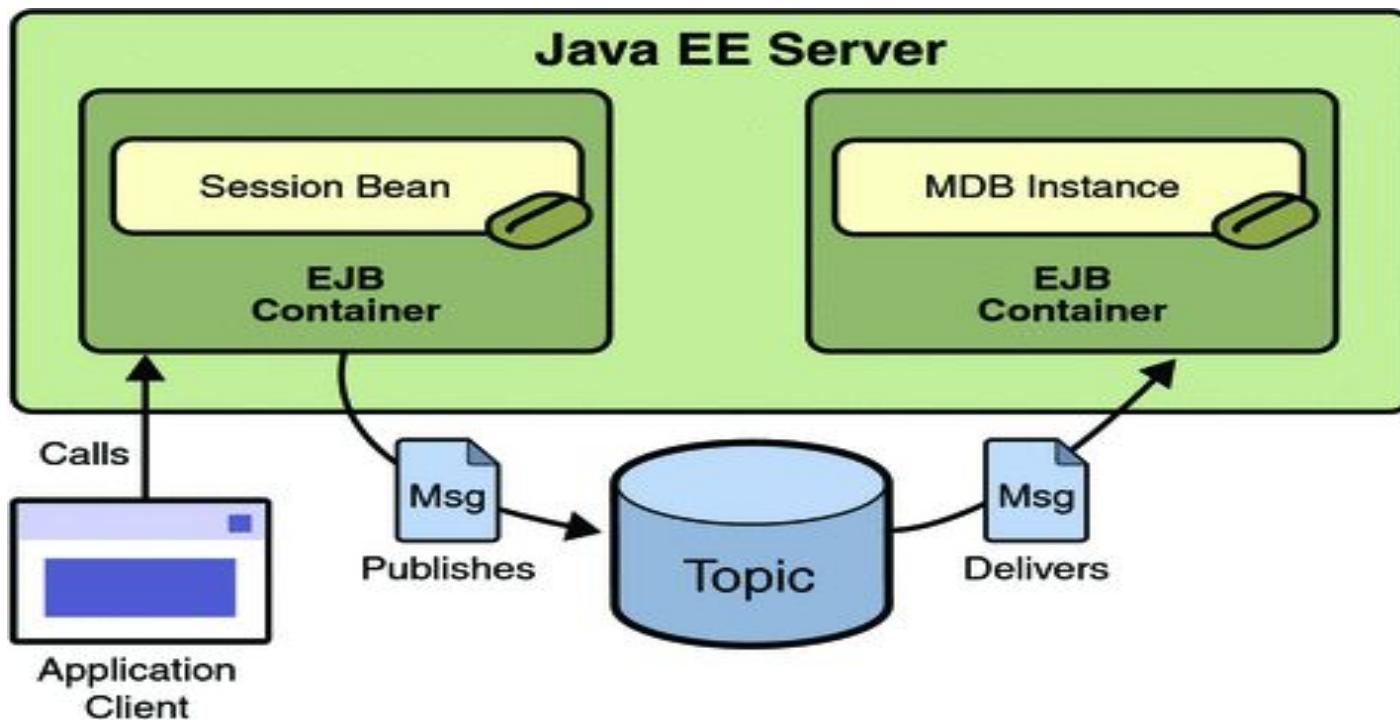
EJB



**Scenario: Remote EJB Client
located in another application**



EJB





EJB Classe

```
package ch.genidea.web.mb;

import ch.genidea.TestBean;

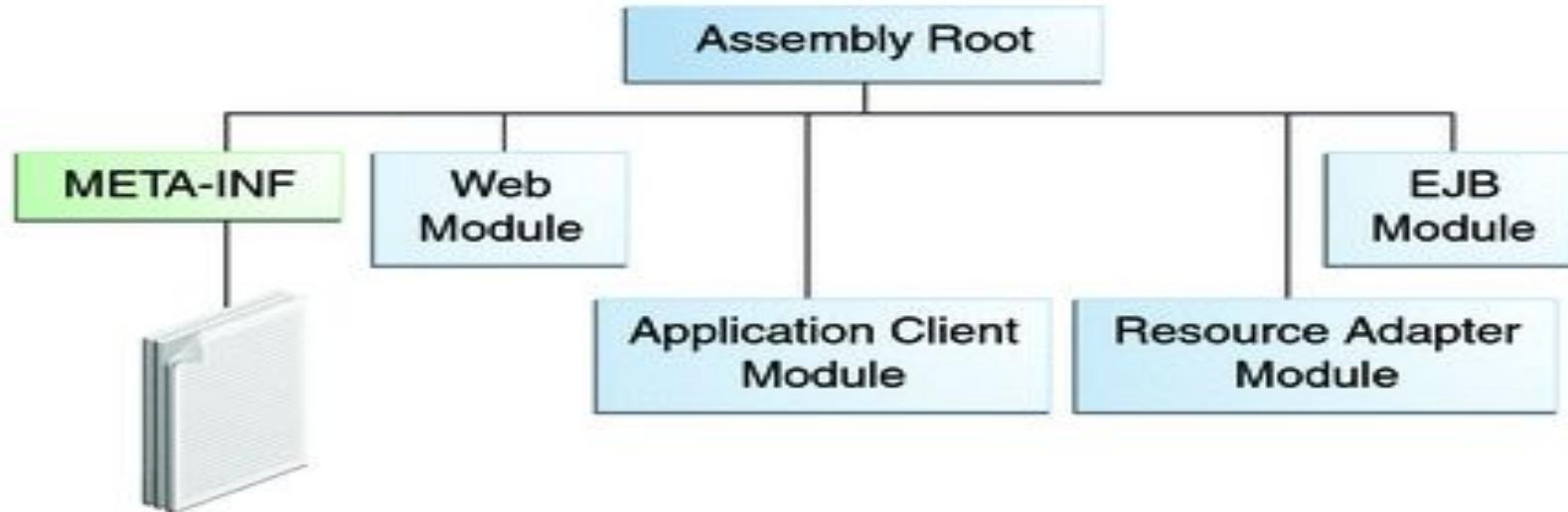
import javax.ejb.EJB;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

@ManagedBean
@SessionScoped
public class HelloMb {
    @EJB
    TestBean testBean;

    public String getText(){
        return testBean.getHello();
    }
}
```



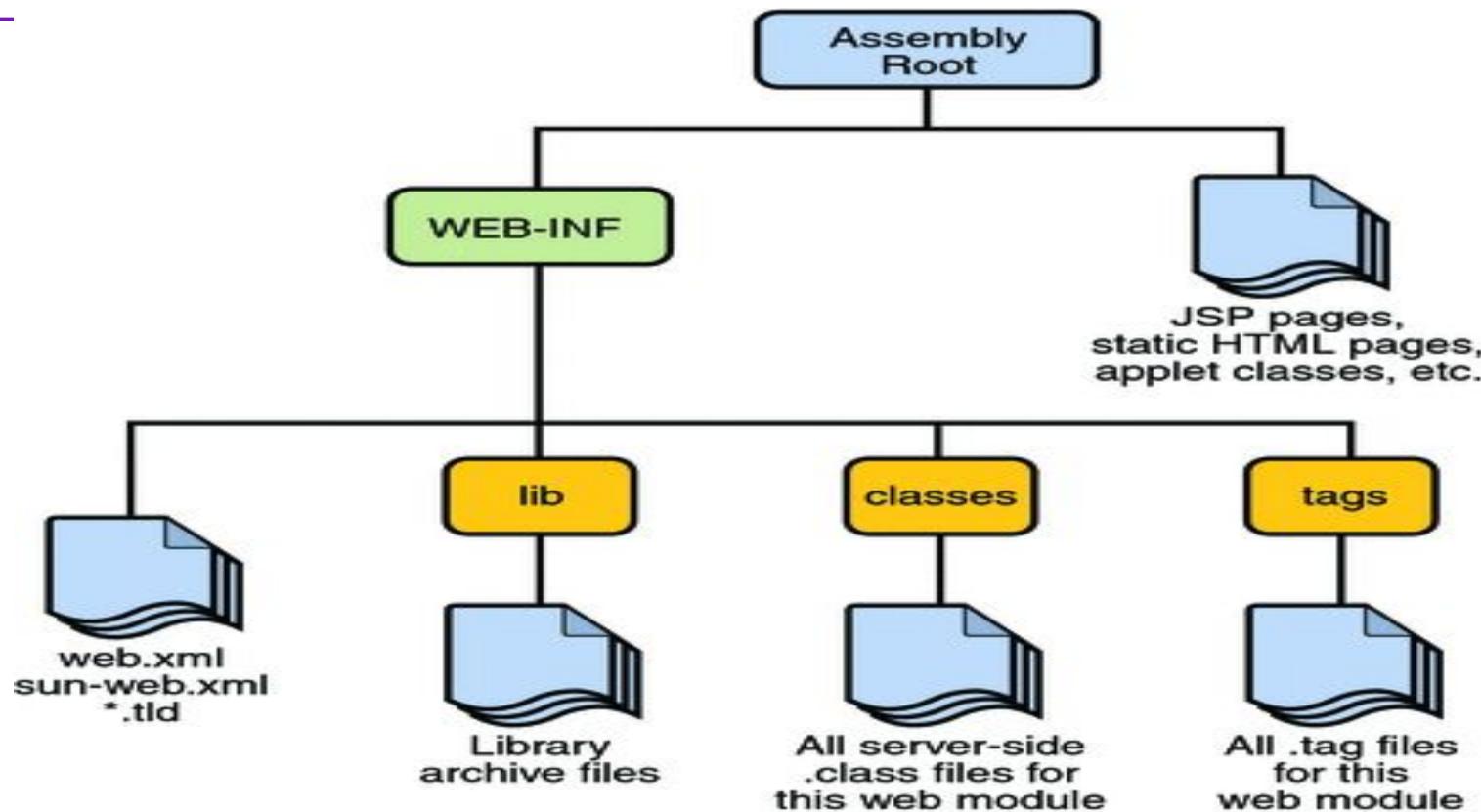
WAR Estrutura



application.xml
glassfish-application.xml
(optional)

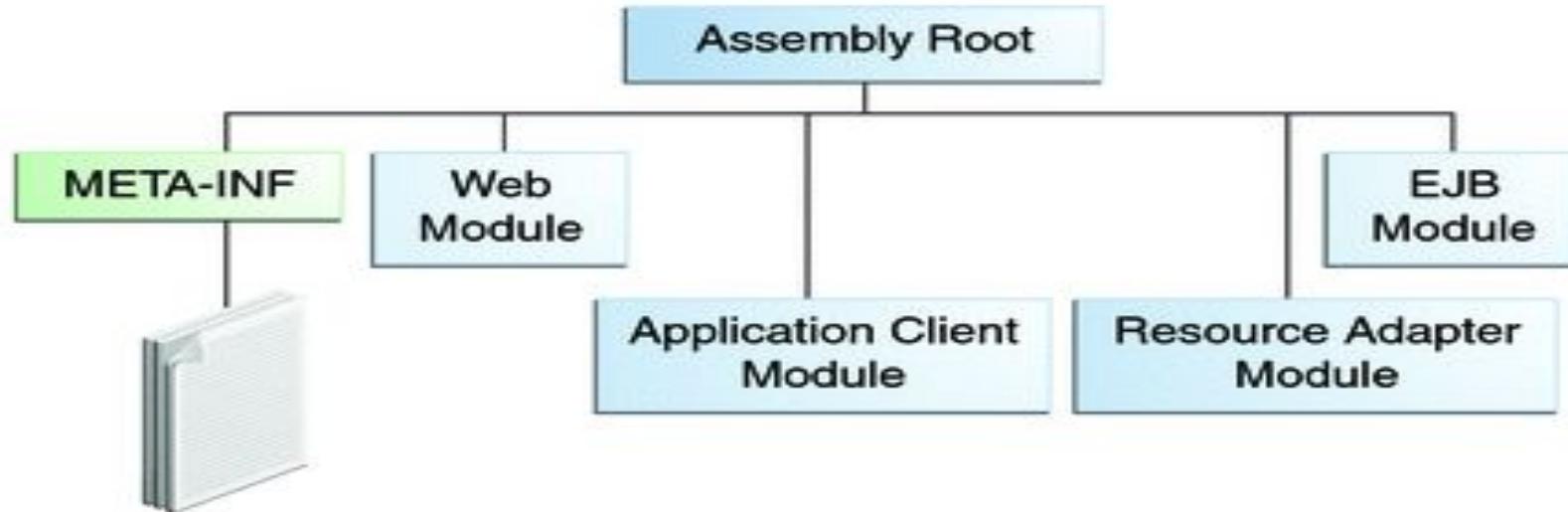


WAR Estrutura





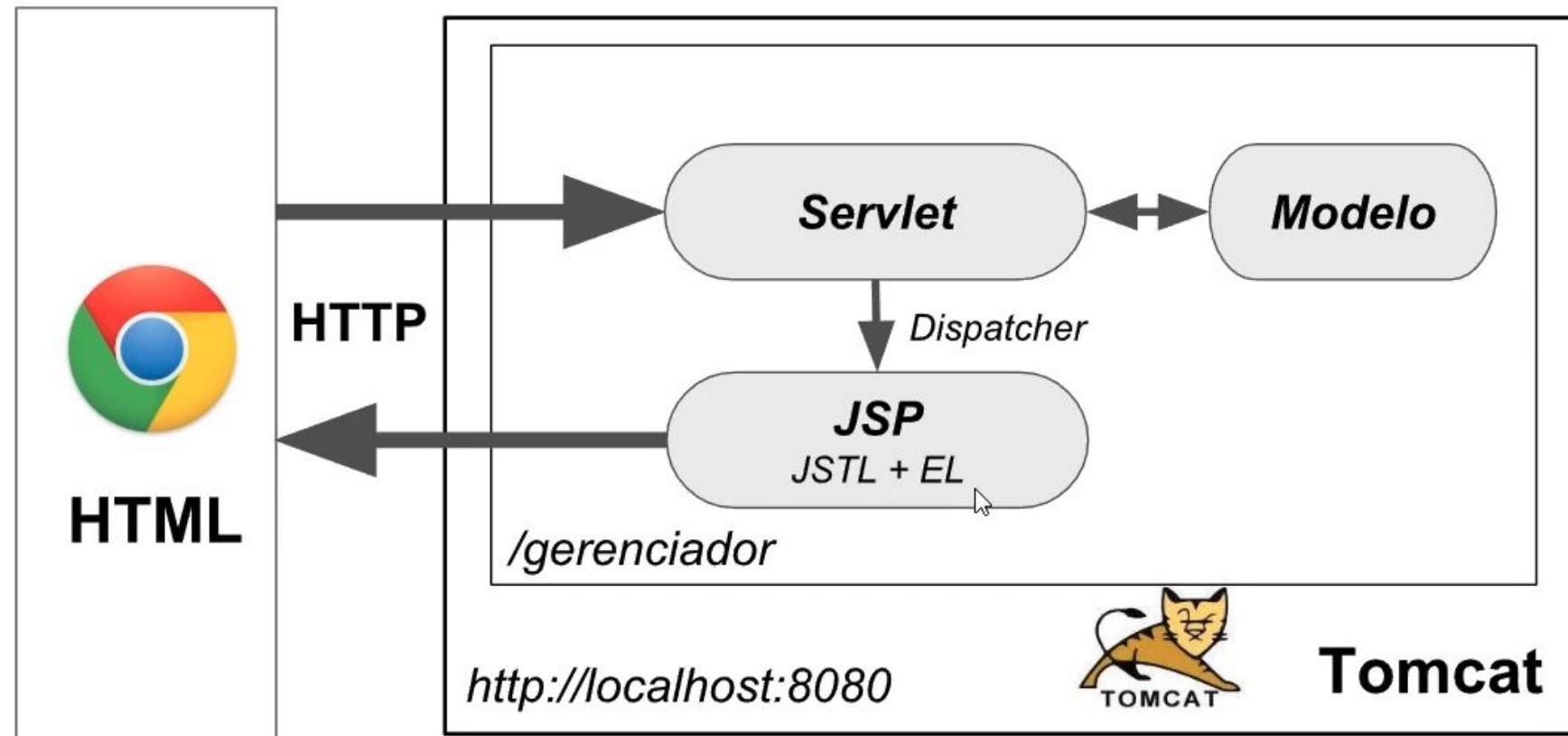
WAR Estrutura



application.xml
glassfish-application.xml
(optional)



WAR Estrutura





WAR Servlet

```
package code;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class HelloServlet
 */
@WebServlet("/HelloServlet")
public class HelloServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response) throws ServletException, IOException {
        response.getWriter().println("Hello friend!");
    }
}
```



WAR Servlet

```
1 public class MeuInicializador implements ServletContainerInitializer {  
2  
3     @Override  
4     public void onStartup (Set<Class<?>>  
5             clazz, ServletContext context) {  
6         ServletRegistration.Dynamic reg =  
7             context.addServlet("MeuServlet",  
8                 "com.exemplo.MeuServlet");  
9         reg.addMapping("/meuServlet");  
10    }  
11 }
```

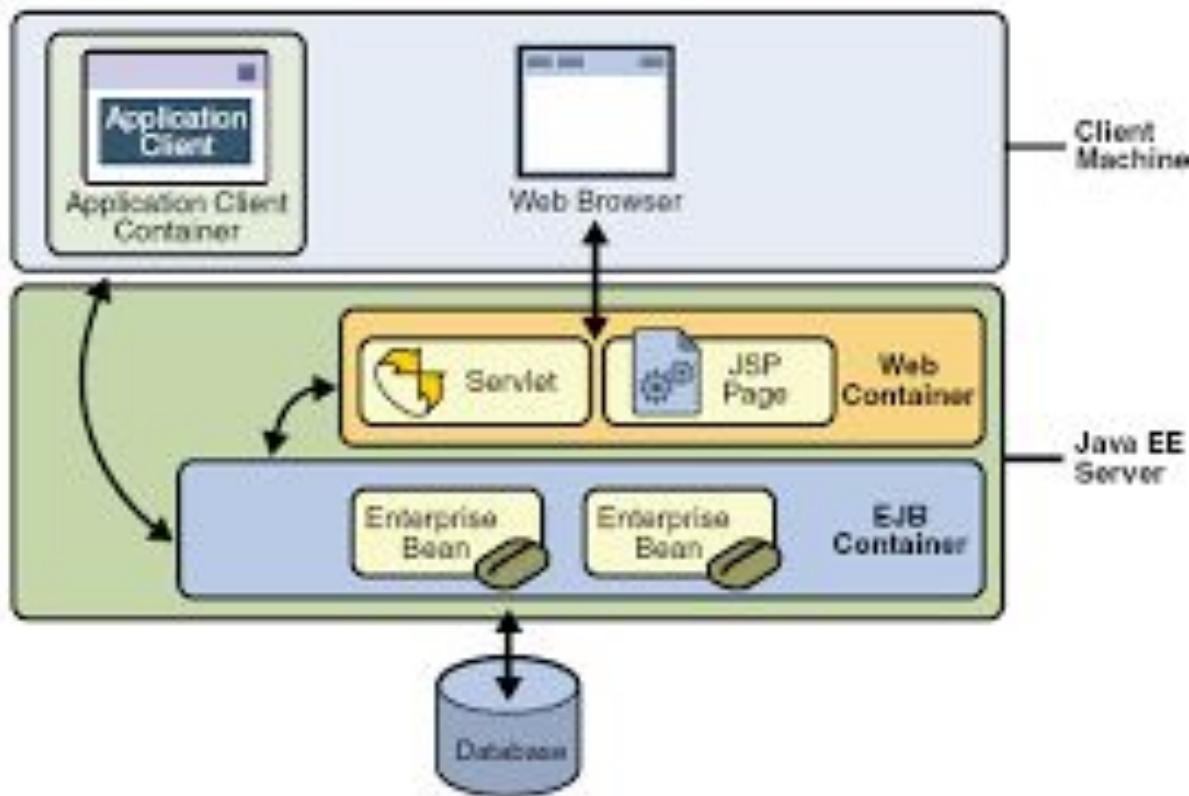


WAR Webservice

```
1 public class WebServiceOla
2 {
3     // código repetido omitido...
4
5     // A anotação @PUT indica que o método a seguir processa requisições HTTP PUT.
6     @PUT
7     // A anotação @Path determina o caminho relativo do método, adicionando
8     // o parâmetro "registro" ao caminho.
9     @Path("enviar/{registro}")
10    @Produces(MediaType.TEXT_PLAIN)
11    public String criaRegistro(@PathParam("registro") String registro)
12    {
13        // Aqui deve ser incluído o código para criar o registro solicitado.
14        return "Registro: \\" + registro + "\\" criado com sucesso!";
15    }
16}
```



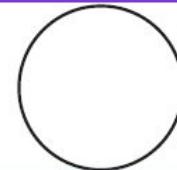
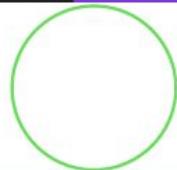
N camada com JavaEE



Spring

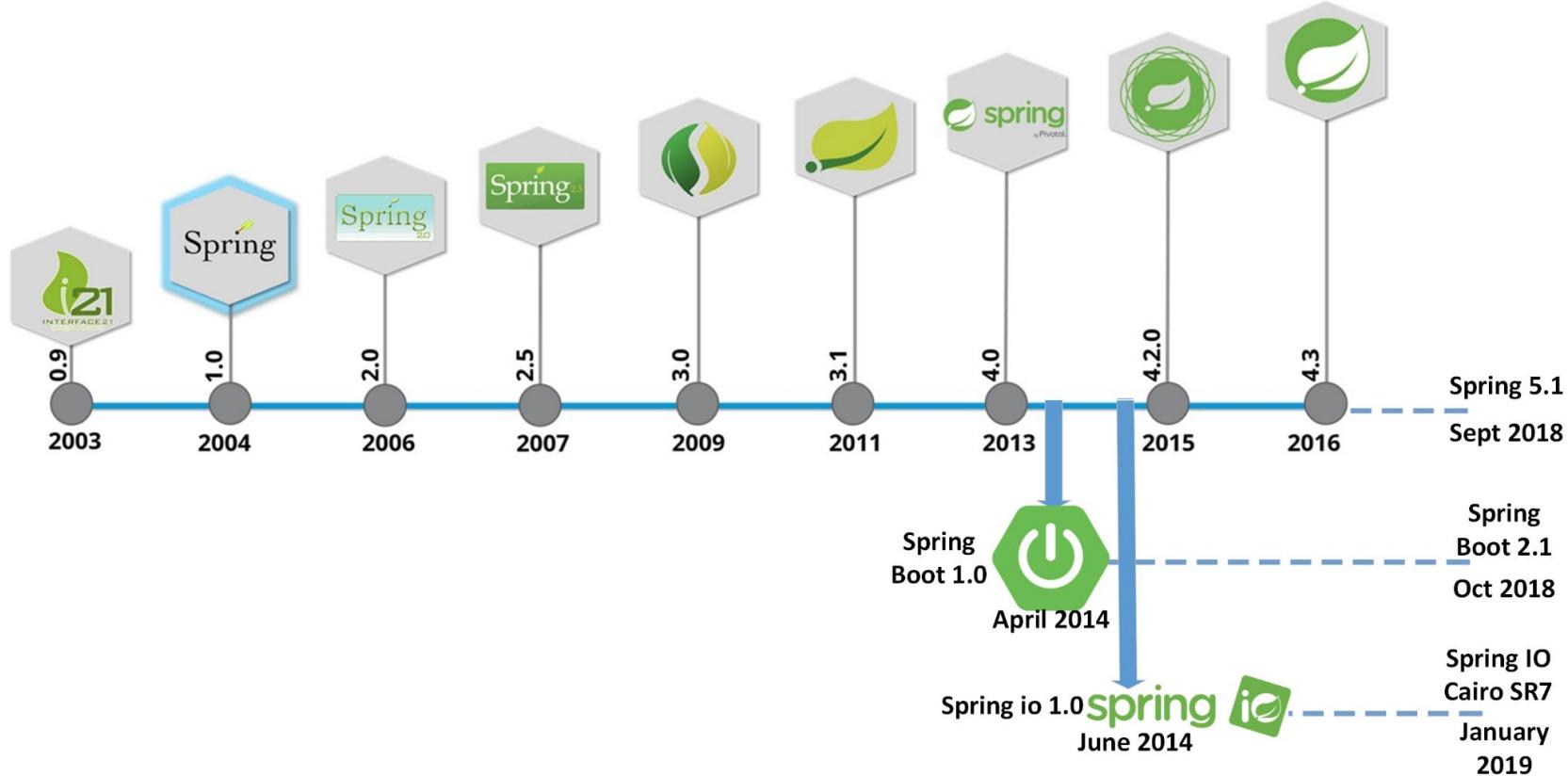


Plataforma Leve



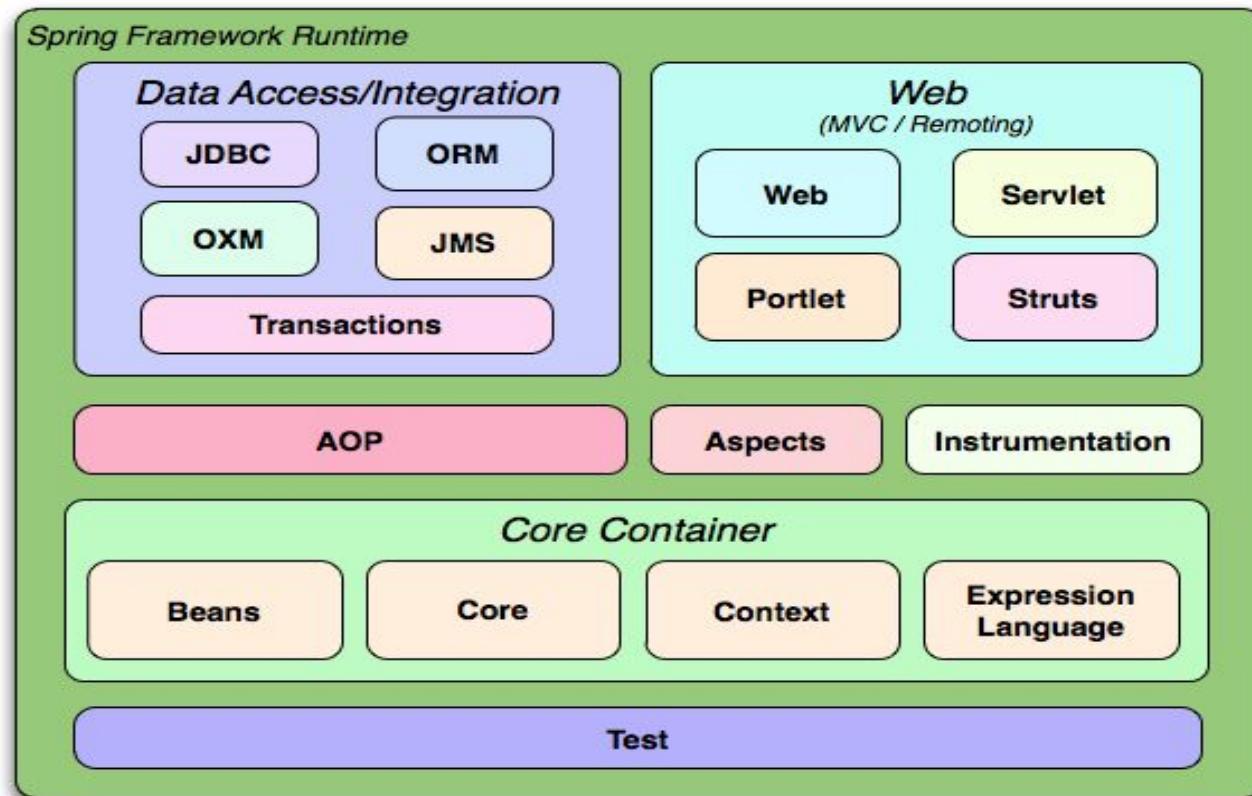


Spring Evolução



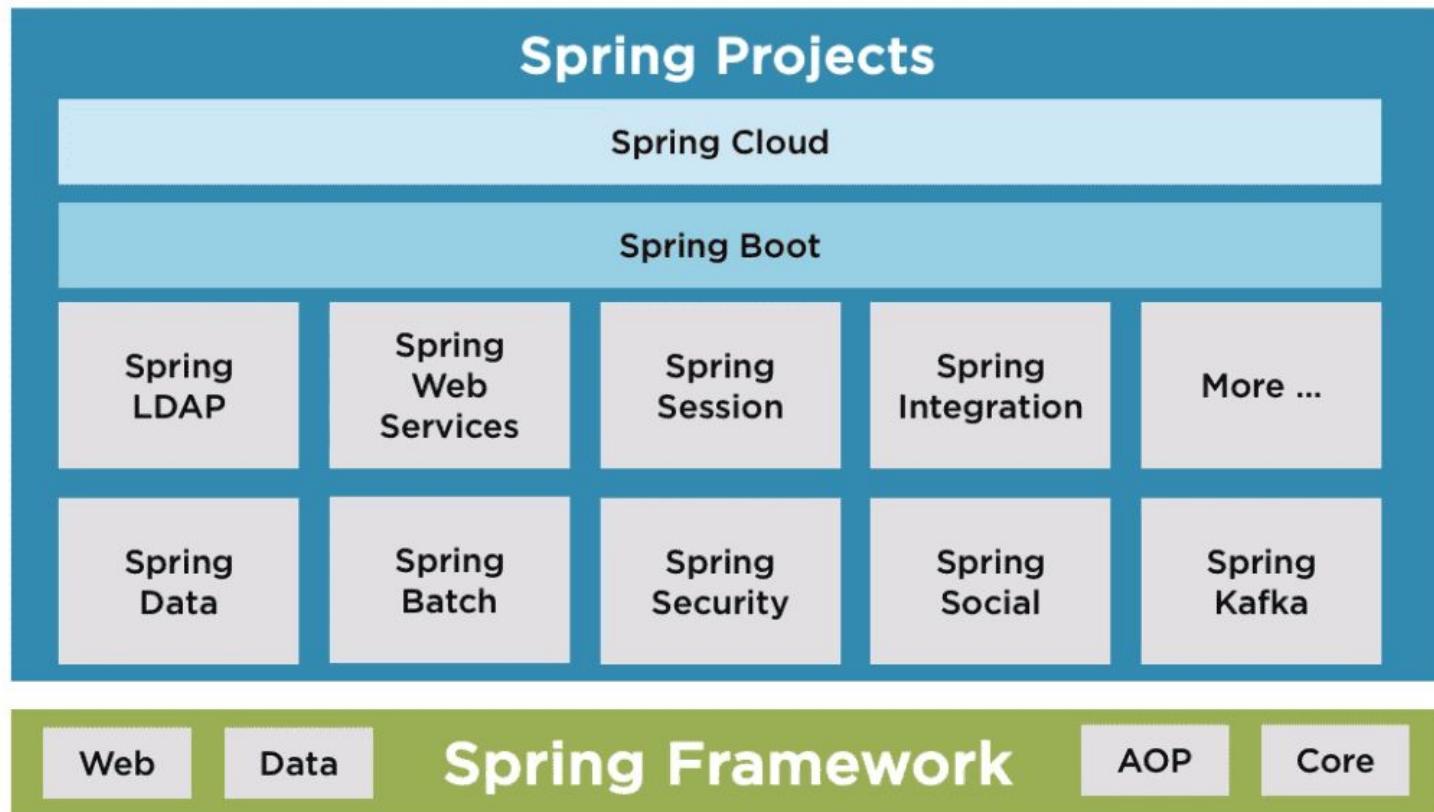


Spring



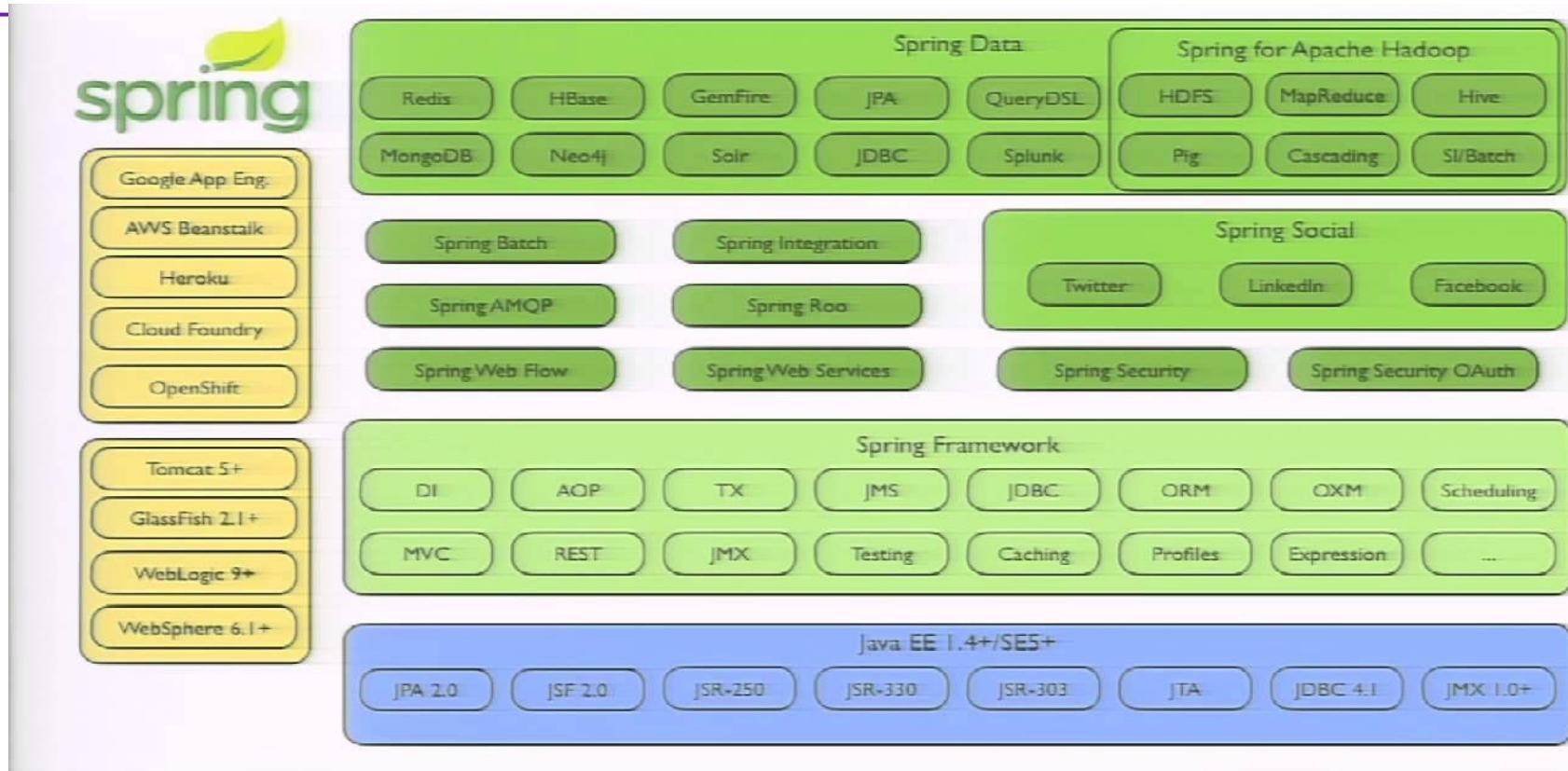


Spring framework stack





Spring framework stack





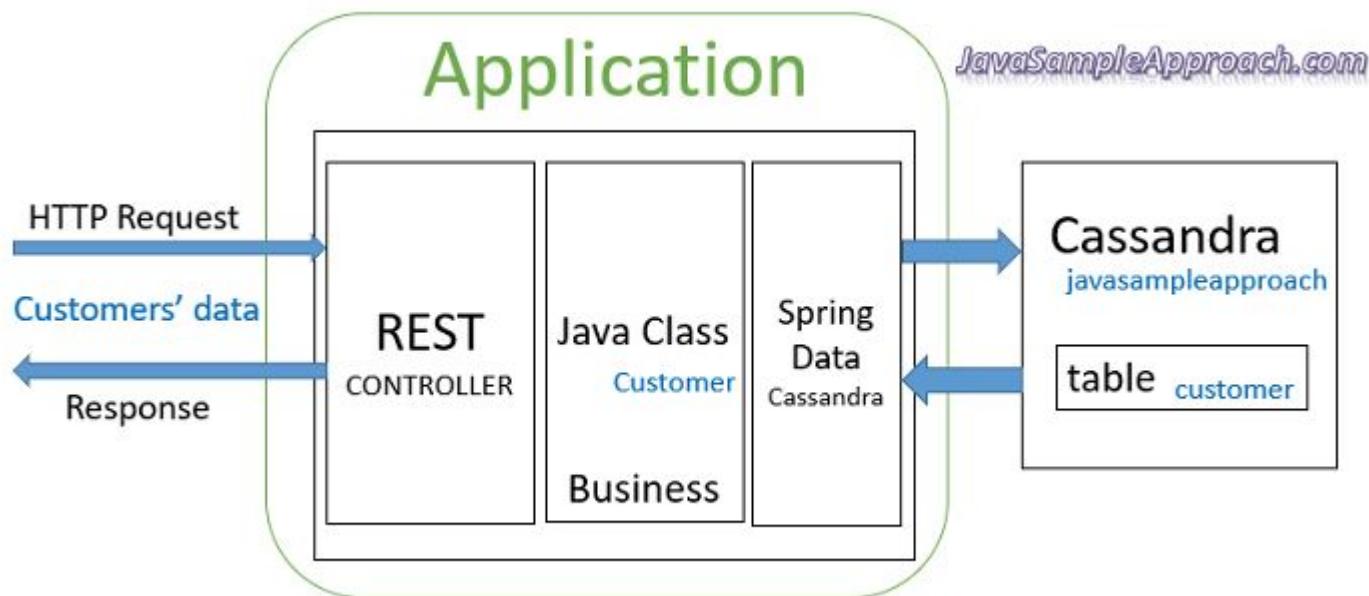
Classe Spring

```
@SpringBootApplication  
public class Springboot2JavaConfigApplication {  
  
    @Bean(name = "emailService")  
    public MessageService emailService() {  
        return new EmailService();  
    }  
  
    @Bean(name = "smsService")  
    public MessageService smsService() {  
        return new SMSService();  
    }  
  
    @Bean(name = "twitterService")  
    public MessageService twitterService() {  
        return new TwitterService();  
    }  
  
    @Bean  
    public MessageProcessor messageProcessor() {  
        return new MessageProcessorImpl(twitterService());  
    }  
}
```

***@SpringBootApplication =
@Configuration +
@EnableAutoConfiguration +
@ComponentScan***

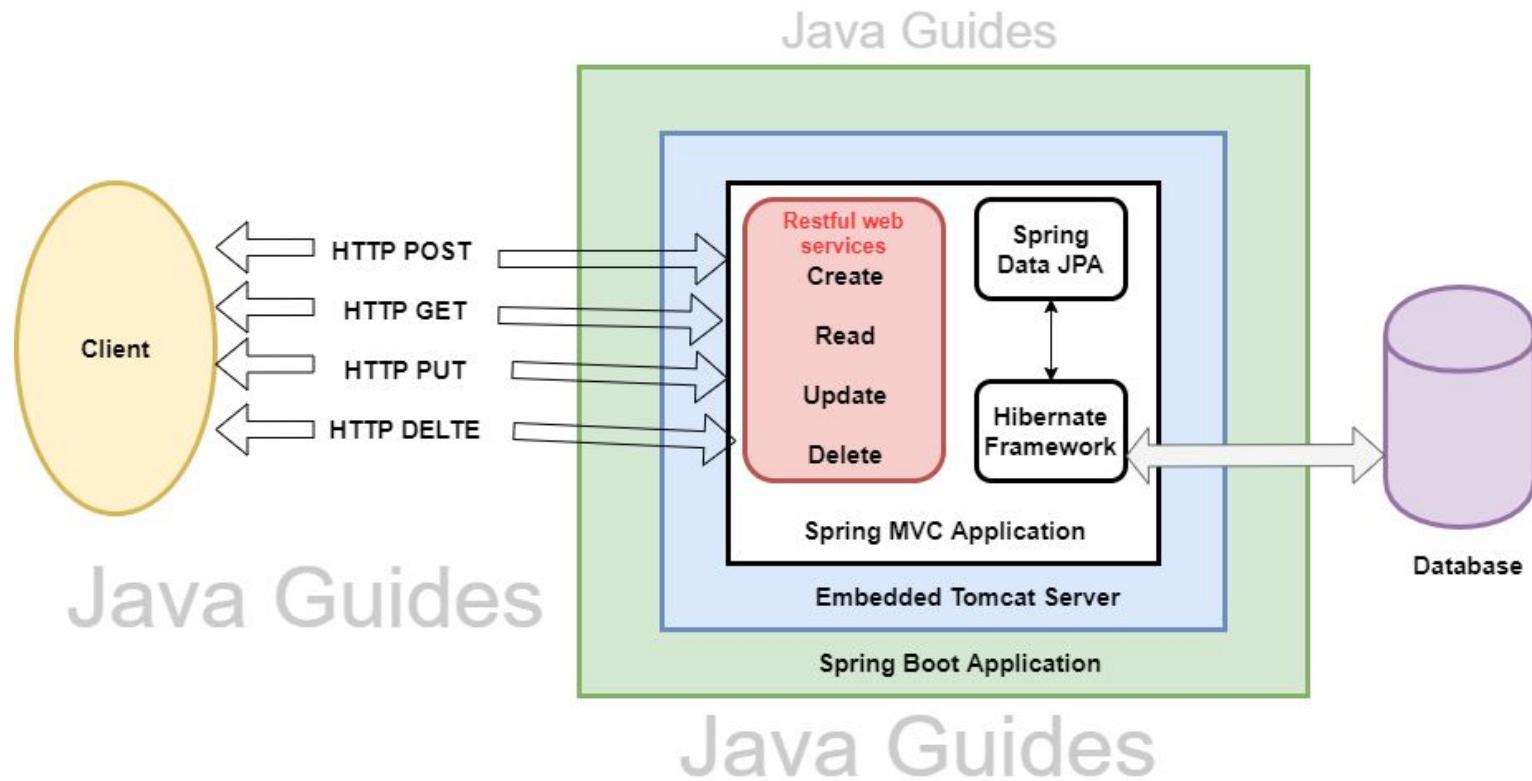


Mundo Spring



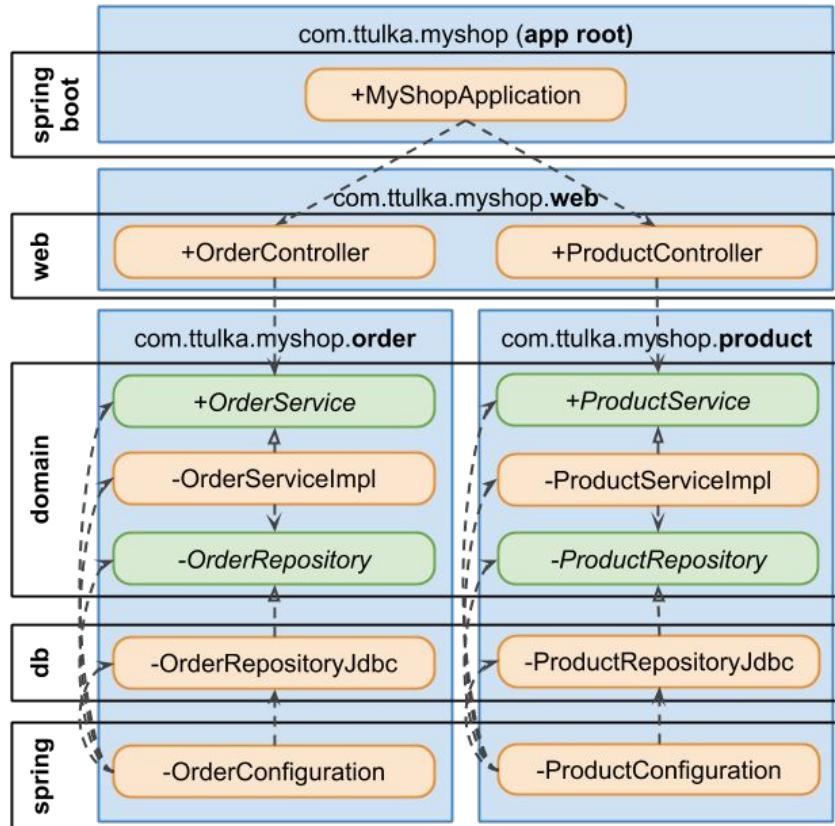


Mundo Spring



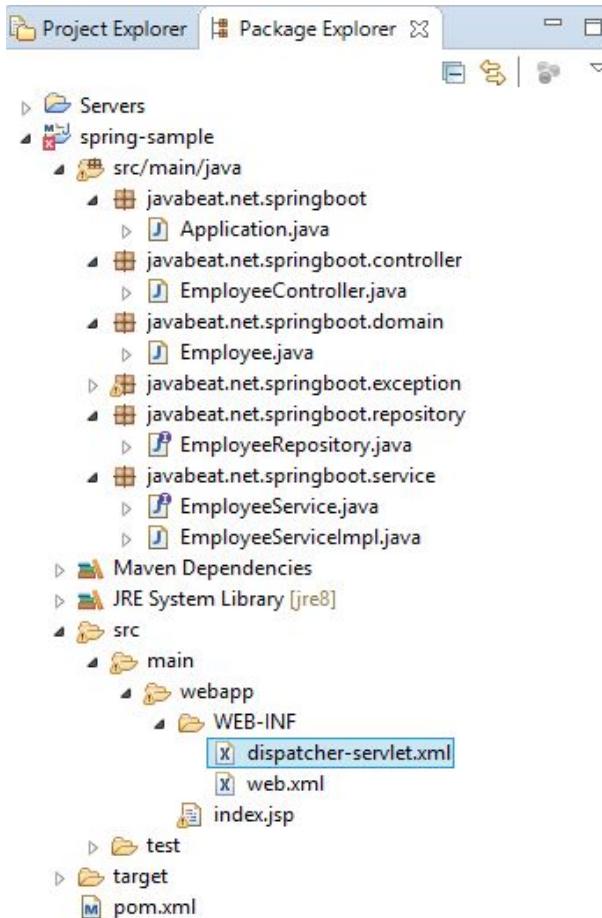


Packaging Spring



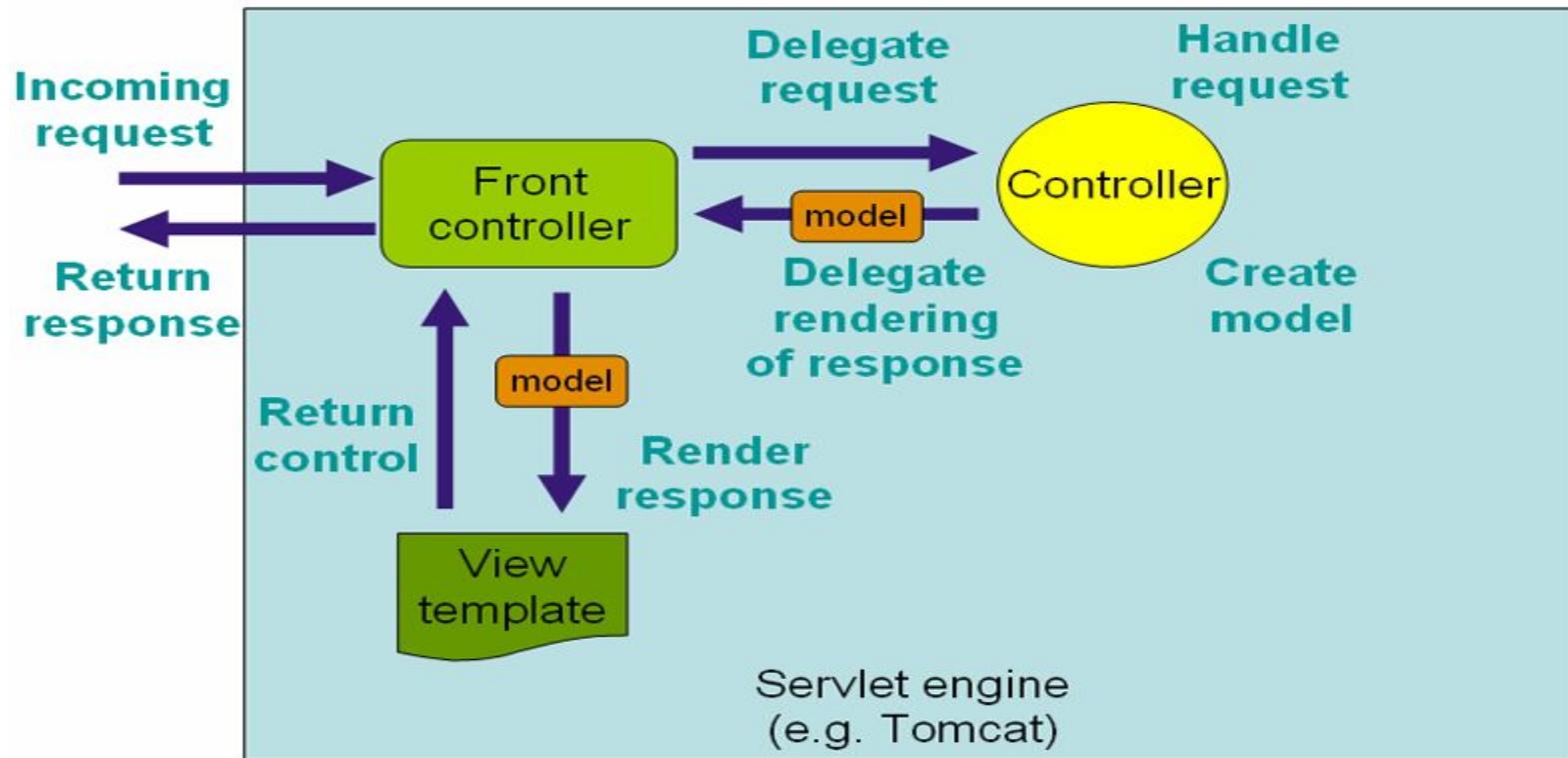


Packaging Spring





Mundo Spring



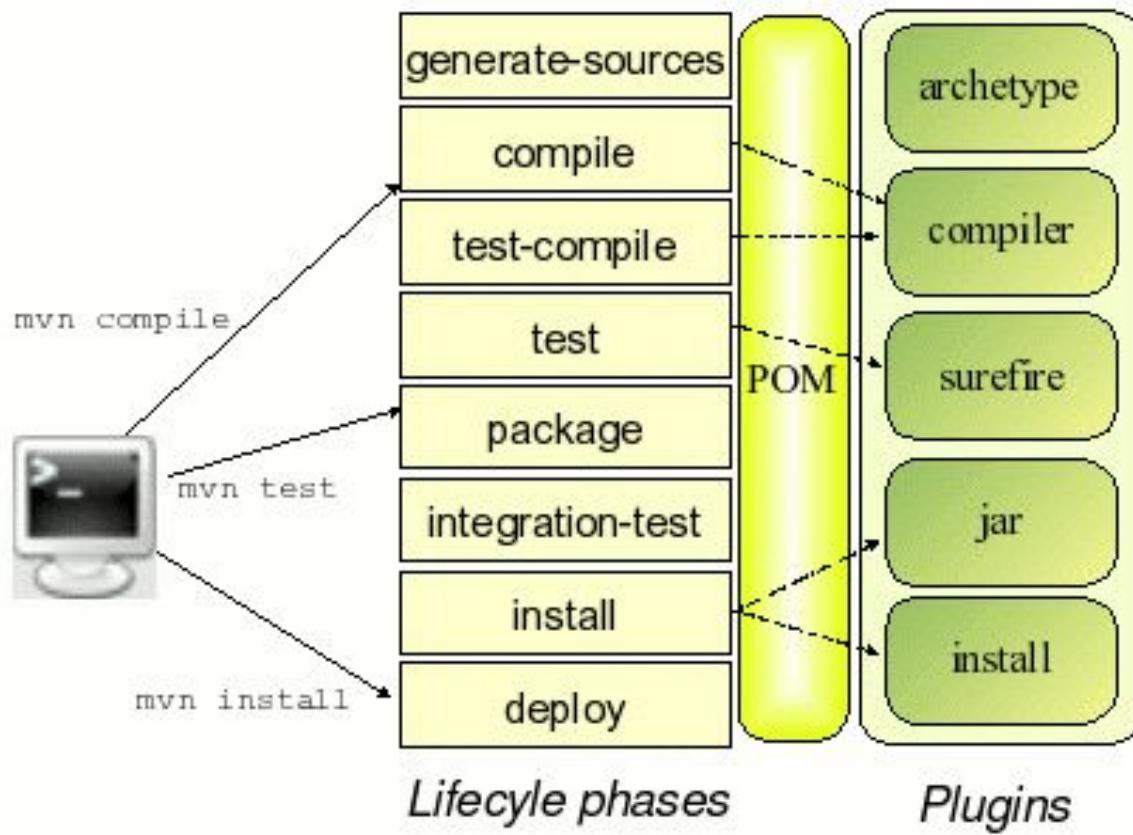
Maven



Gerenciamento de pacote



Maven



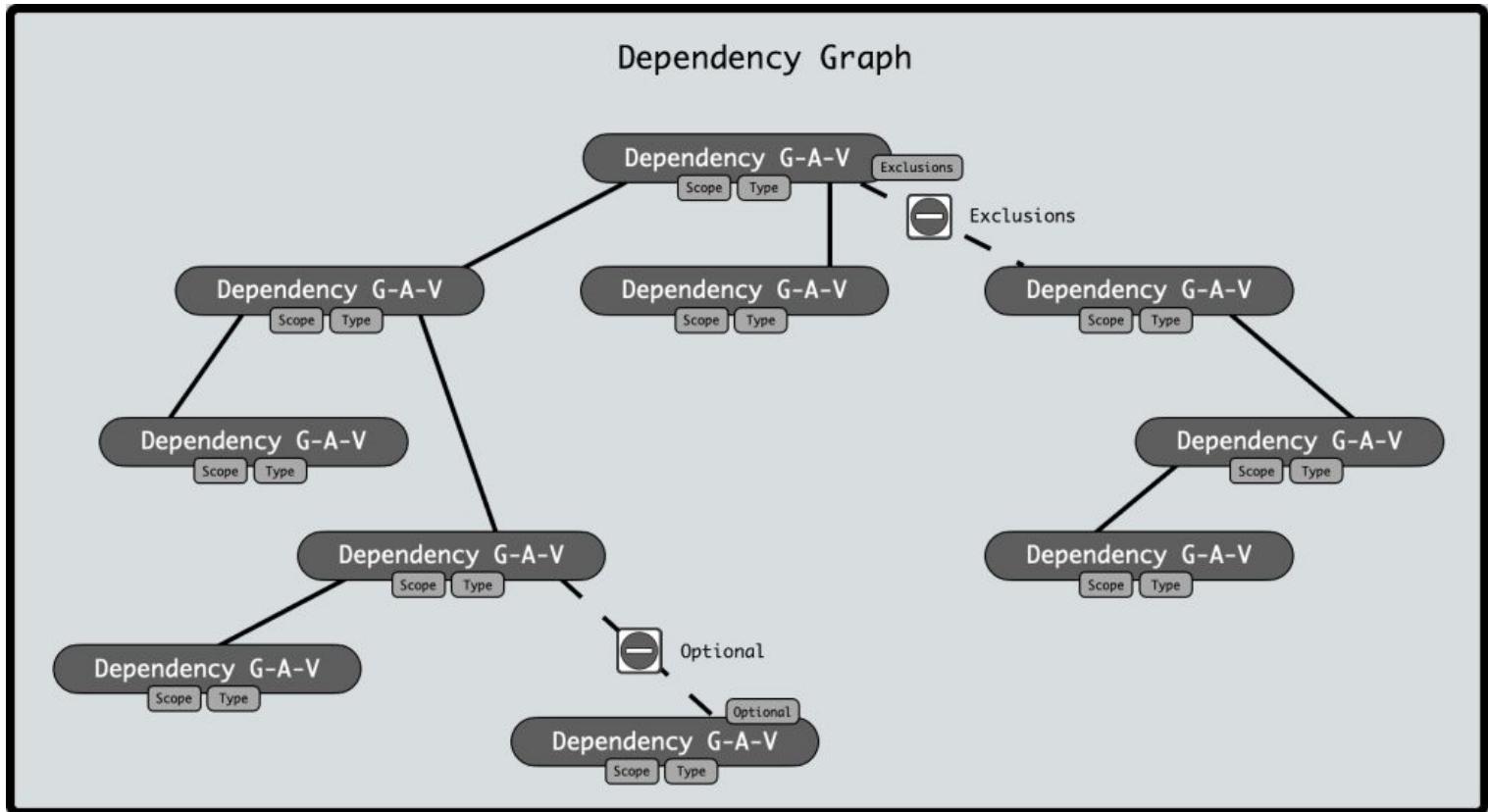


Maven dependência

```
<project>
  ...
  <dependencies>
    <dependency>
      <groupId>a.group-id</groupId>
      <artifactId>an-artifact</artifactId>
      <version>1.0</version>
      <exclusions>
        <exclusion>
          <groupId>transitive.group-id</groupId>
          <artifactId>excluded-artifact</artifactId>
        </exclusion>
      </exclusions>
      <optional>true</optional>
    </dependency>
    <dependency>
      <groupId>another.group.id</groupId>
      <artifactId>another-artifact</artifactId>
      <version>1.0.0-SNAPSHOT</version>
      <type>zip</type>
      <scope>runtime</scope>
    </dependency>
  </dependencies>
</project>
```

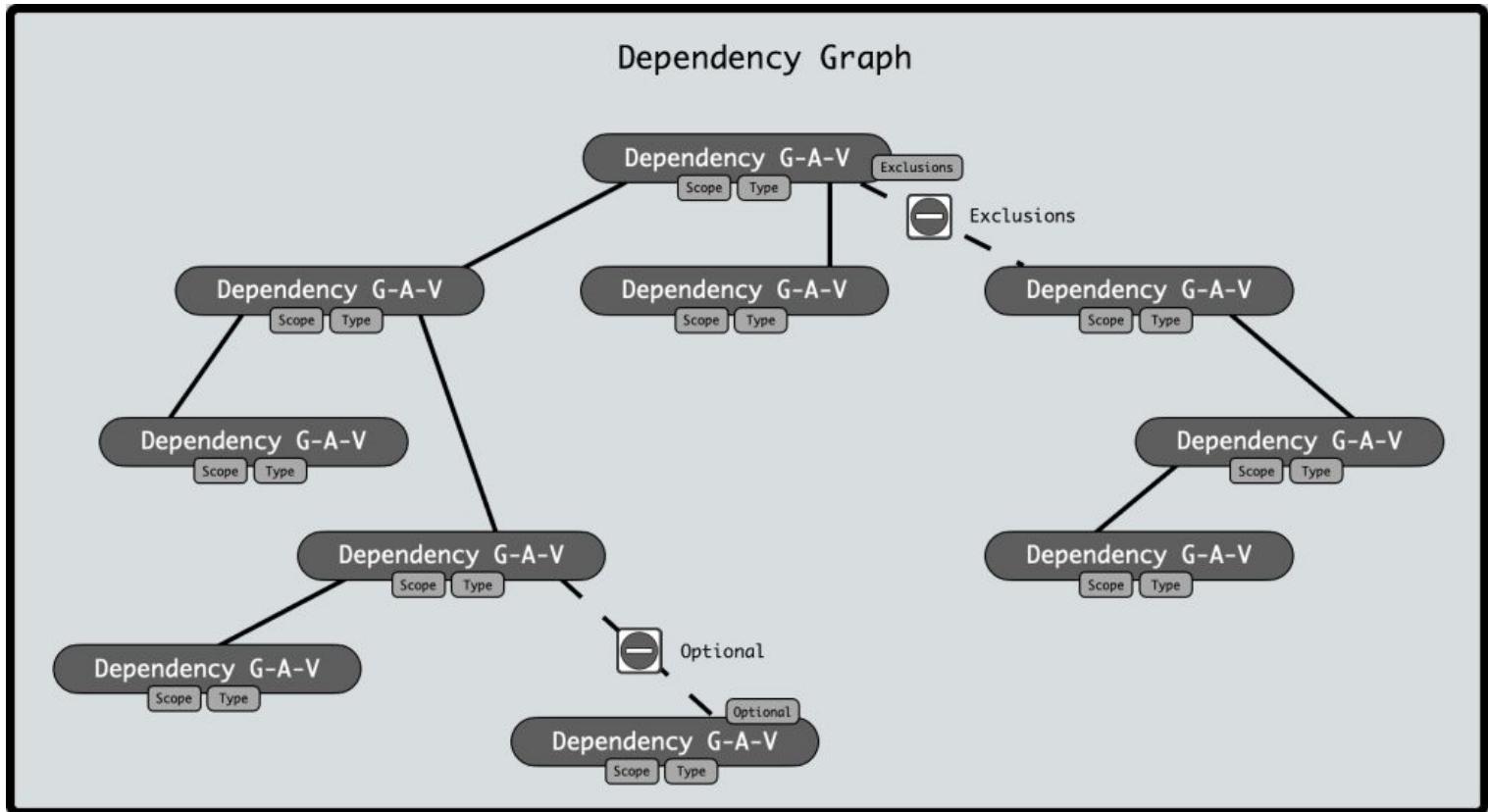


Maven dependência





Maven dependência





Encerramos por hoje !!!

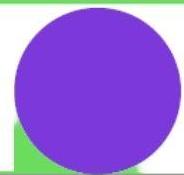


Monolíticos Vs. Microservices

Decisão arquitetura

Monolítico

Único





Definição

monolítico

1. relativo ou semelhante a monólito
2. feito de uma só pedra
3. *figurado* diz-se de um todo unido, inseparável e homogêneo

meudicionario.org



Monolítico - Prós

Dados são compartilhados e fáceis de serem consumidos

Único deploy

Maior controle transacional

Menor número de integrações

Um sistema monolítico é mais fácil de entender, possui uma curva de aprendizado menor e, obviamente, existem mais desenvolvedores familiarizados com este modelo.

O ambiente de desenvolvimento fica muito mais simples quando a arquitetura é monolítica.



Monolítico - Contras

Difícil de escalar: a medida que a aplicação vai crescendo a tendência é sempre duplicar a infraestrutura e colocar os servidores embaixo de um *load balancer* ou simplesmente aumentar o tamanho do servidor.

Por ser construído como um único código, se algo quebrar todo o sistema fica indisponível.

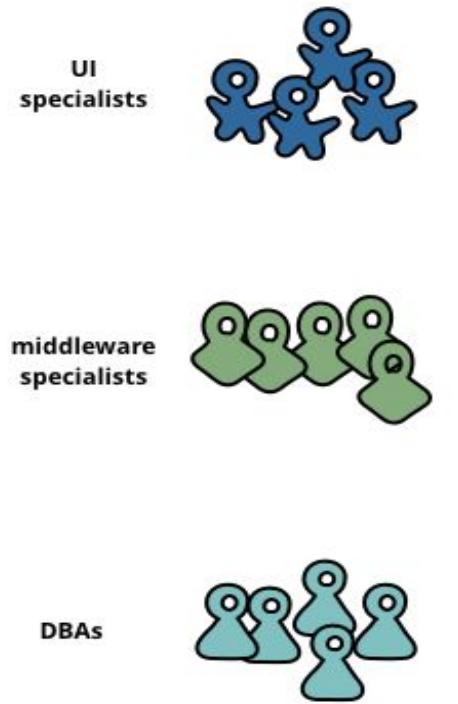
Ao longo do tempo o sistema vai crescendo, se tornando mais complexo e consumindo cada vez mais recursos

Dificuldade para colocar alterações em produção

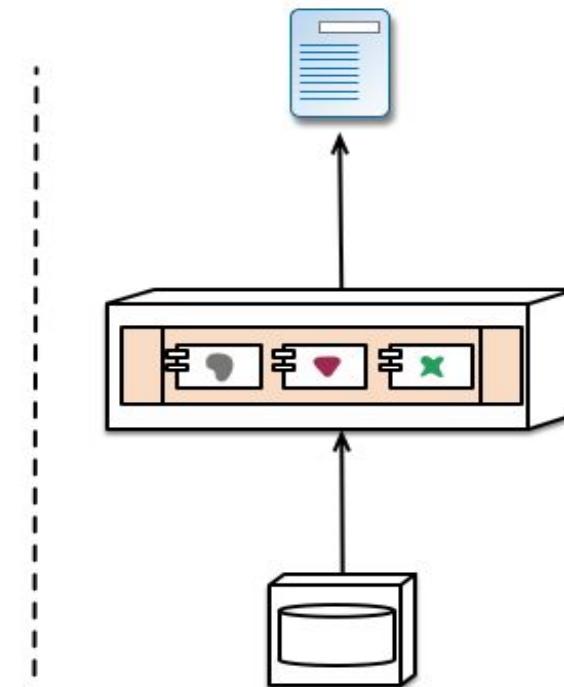
Qualquer mudança, por menor que seja, requer a reinicialização do sistema. Como isso pode causar algum risco operacional, é necessário que as equipes de desenvolvimento, testes e manutenção desses sistemas acompanhem essas alterações



Atuação do time



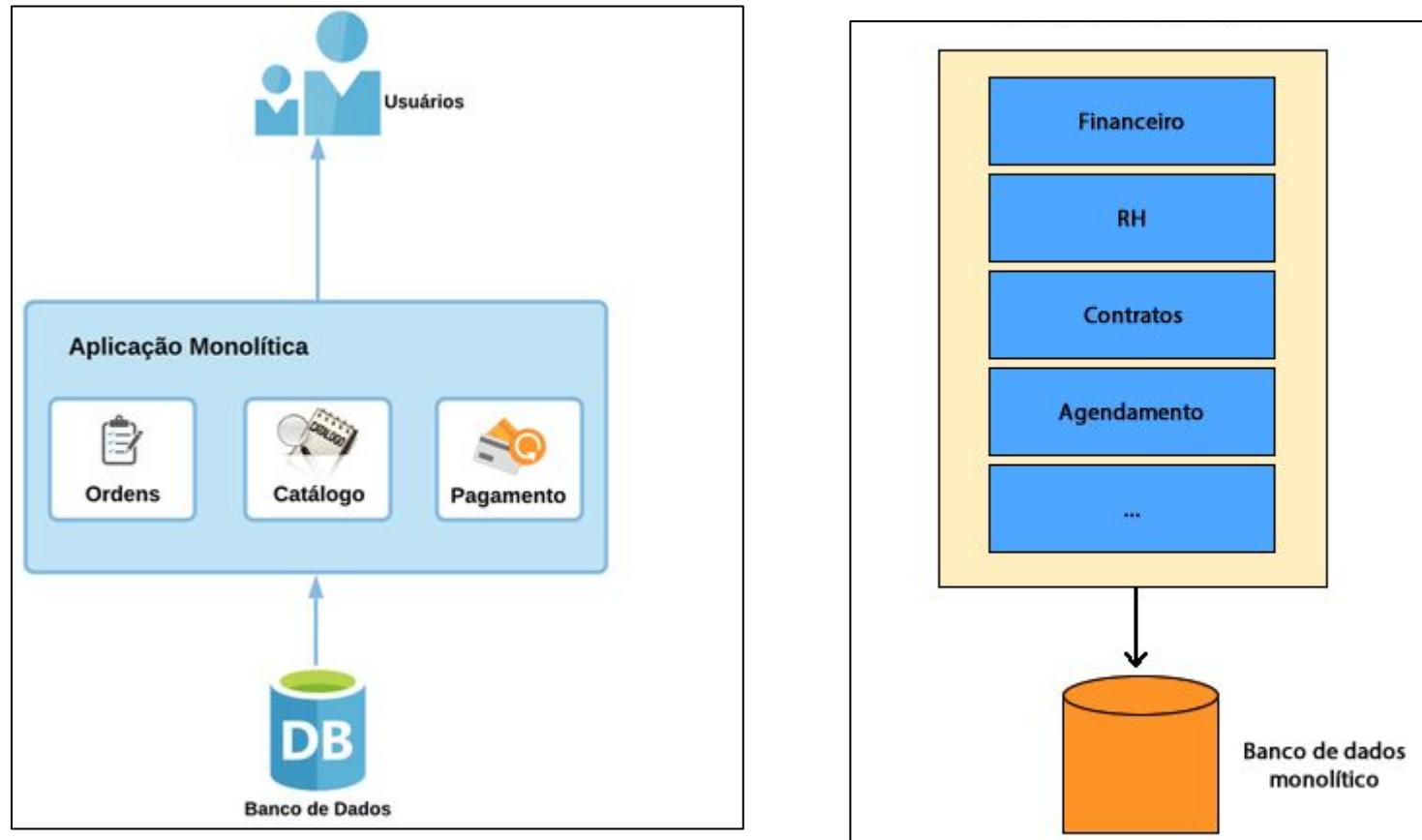
Siloed functional teams...



... lead to siloed application architectures.
Because Conway's Law

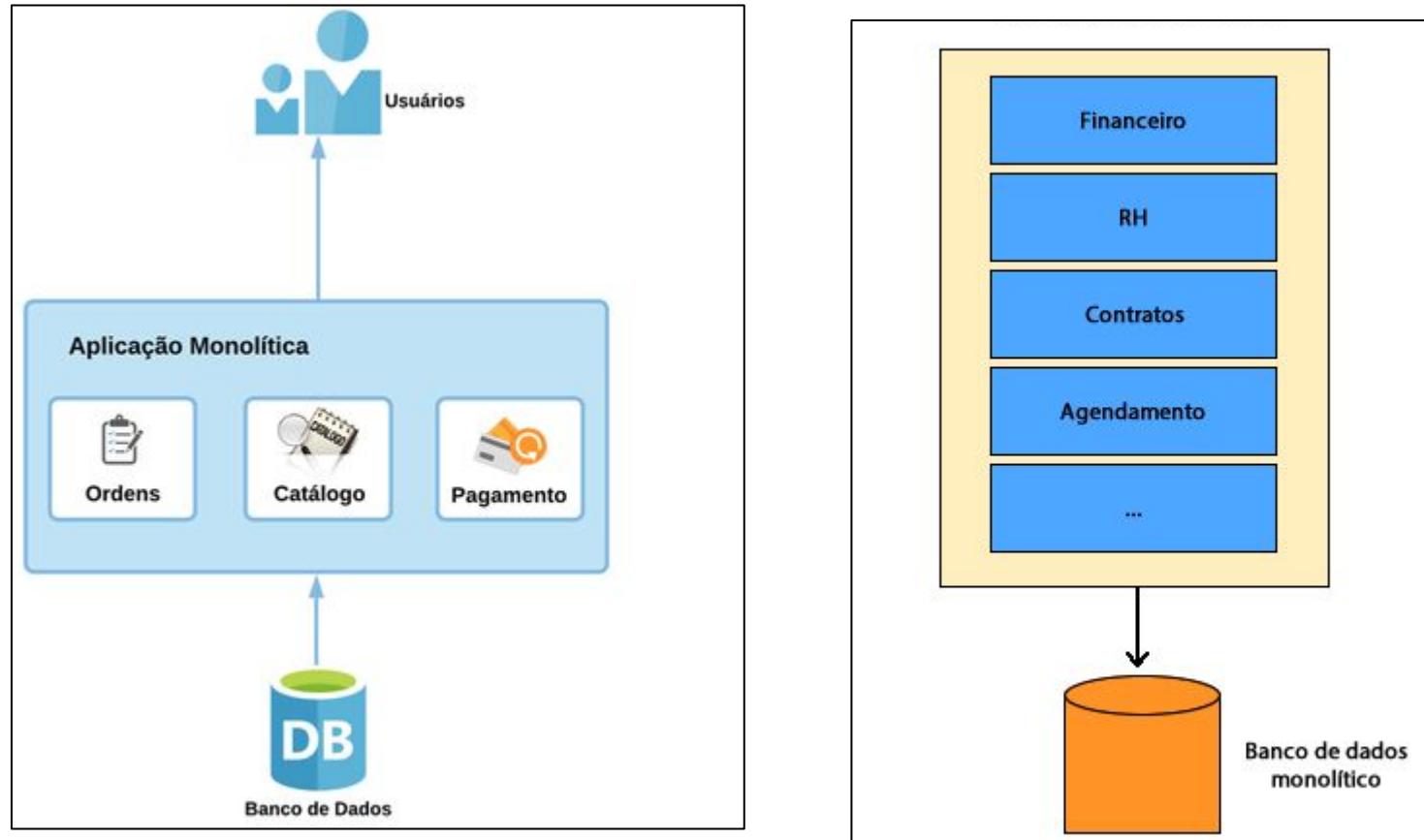


Modelo



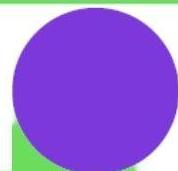


Modelo



Microsservices

Vários



Microsserviços

Prós e contras



Microsserviço - Prós

Arquiteturas de microsserviços são tipicamente melhor organizadas. Cada microsserviço tem um trabalho muito específico, bem definido e serve a um único propósito.

Eles permitem *deployment* independente, ou seja, caso você precise subir uma nova versão de um serviço, isso não afetará a disponibilidade dos outros.

É possível isolar serviços importantes em uma infraestrutura apartada e escalá-los independentemente do restante do sistema.

Os microsserviços permitem o desenvolvimento paralelo, numa mesma equipe ou separando as demandas em times distintos.

Permite independência tecnológica em cada um dos serviços. Por exemplo, numa API de pesquisa podemos utilizar C++ ou Rust para otimizar a performance ao máximo, enquanto os outros serviços podem ser desenvolvidos numa linguagem mais produtiva e simples como C# ou Java, por exemplo.



Microsserviço - Contra

Você vai precisar de *API Gateway* para orquestrar os microsserviços.

É impossível ser produtivo se não houver uma boa automatização de *deploy*, pois os microsserviços são freqüentemente implantados em suas próprias máquinas virtuais ou contêineres, o que é muito trabalhoso para gerenciar manualmente.

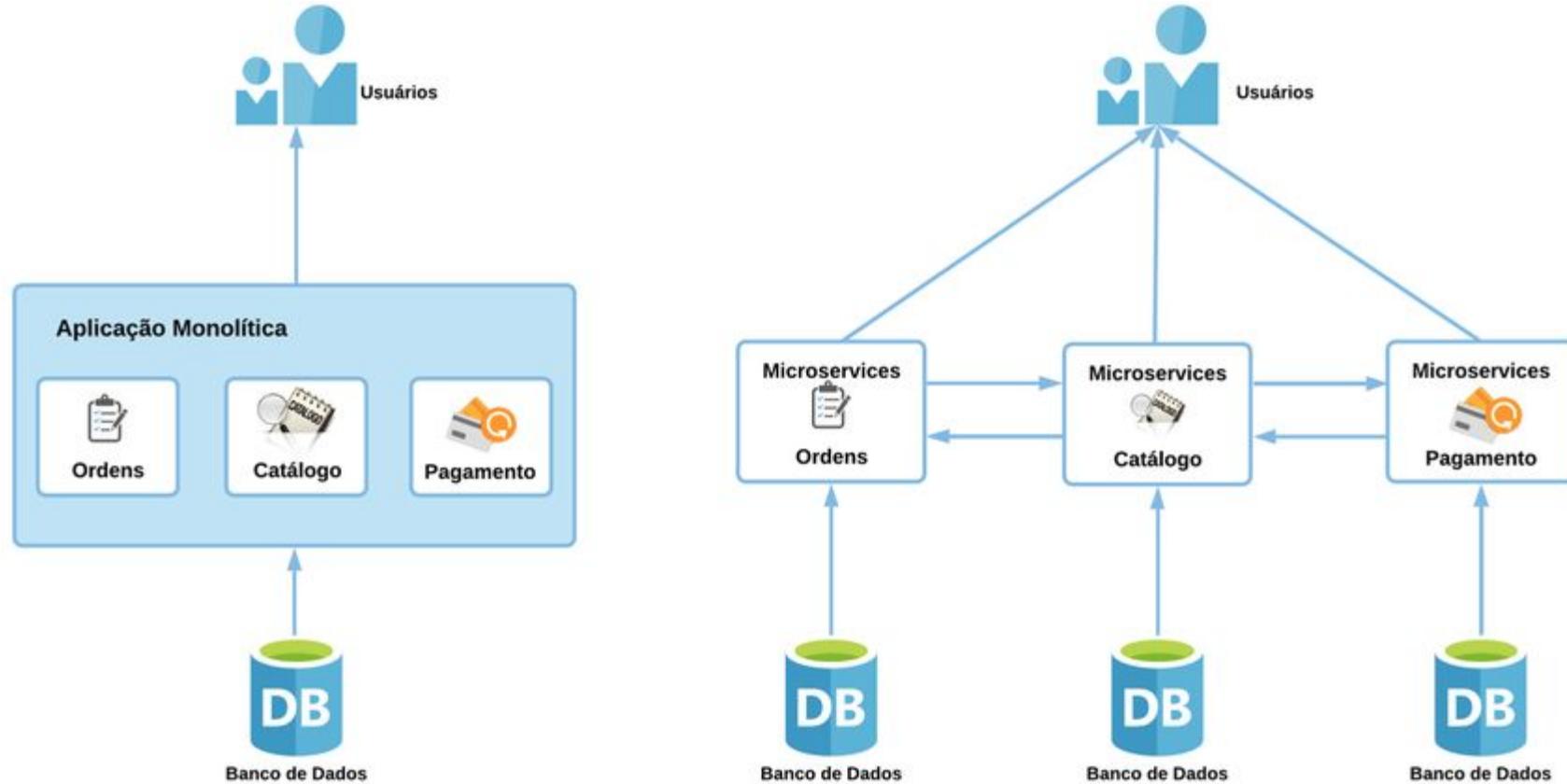
É mais difícil de criar uma camada para encapsular preocupações transversais, comuns a vários módulos numa arquitetura de microsserviços.

Pode se tornar muito complexo se o ambiente não possui uma boa observabilidade e monitoria.

Necessário o conceito SAGA para gerenciar contexto de transações entre eles.

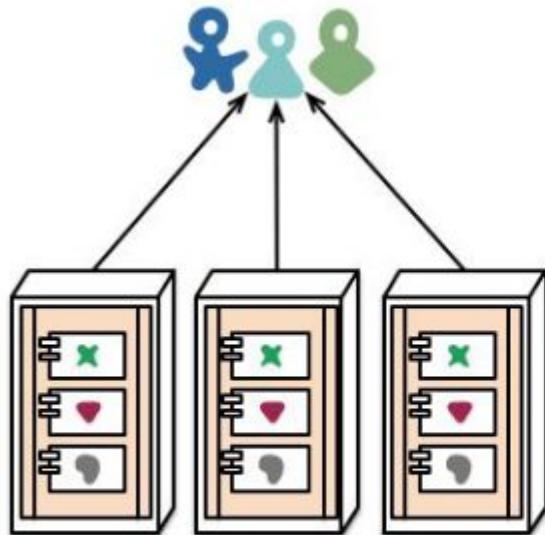


Arquitetura

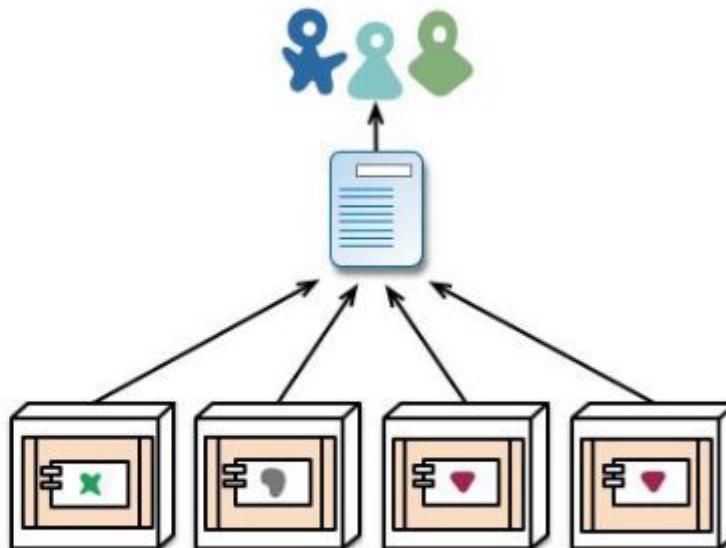




Arquitetura



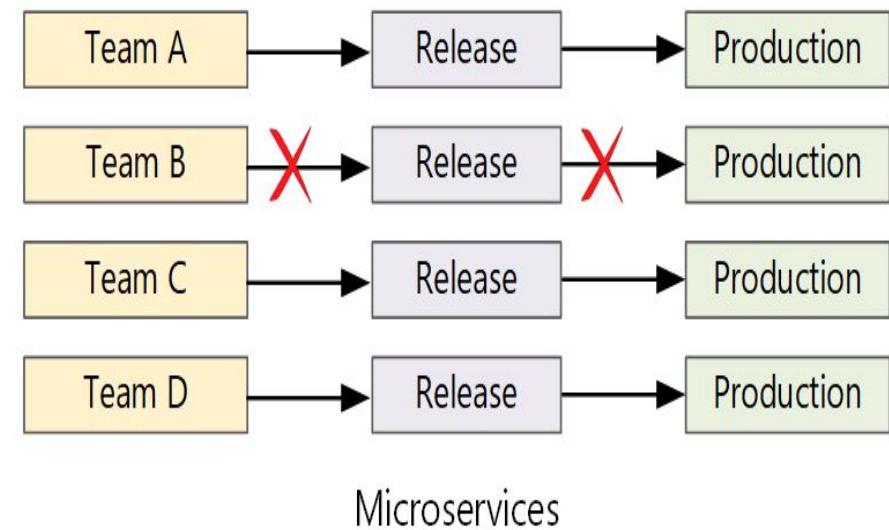
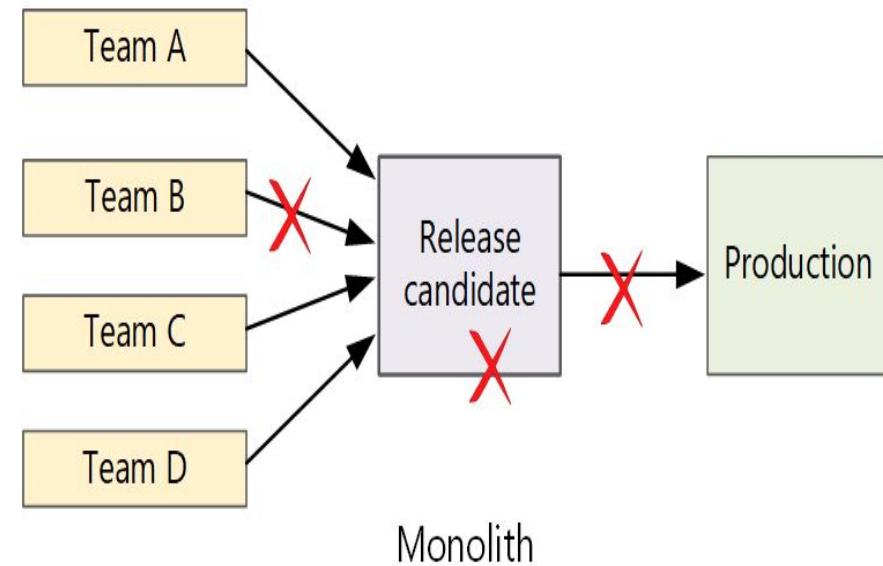
Ambiente monolítico - múltiplos módulos no mesmo processo



Microsserviços - módulos rodando em diferentes processos

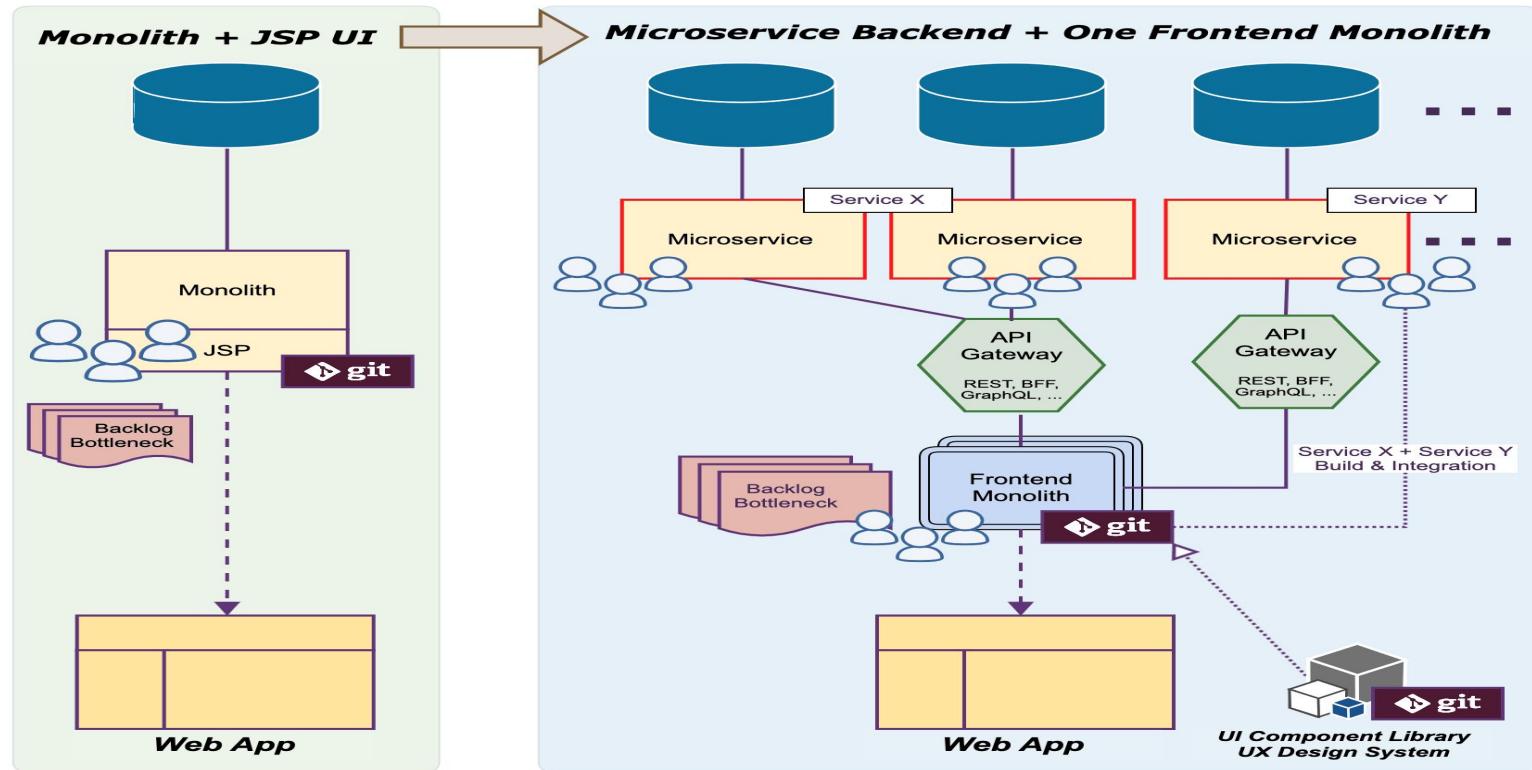


Times atuando na evolução



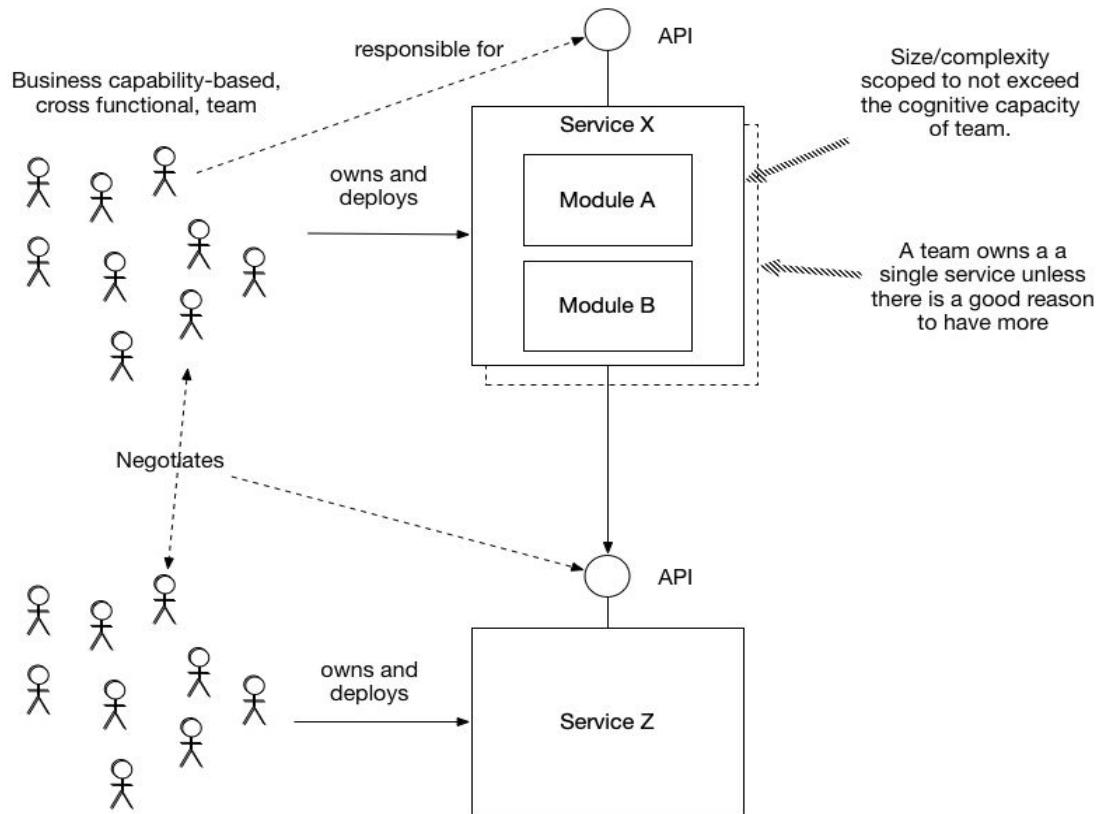


Times atuando na evolução



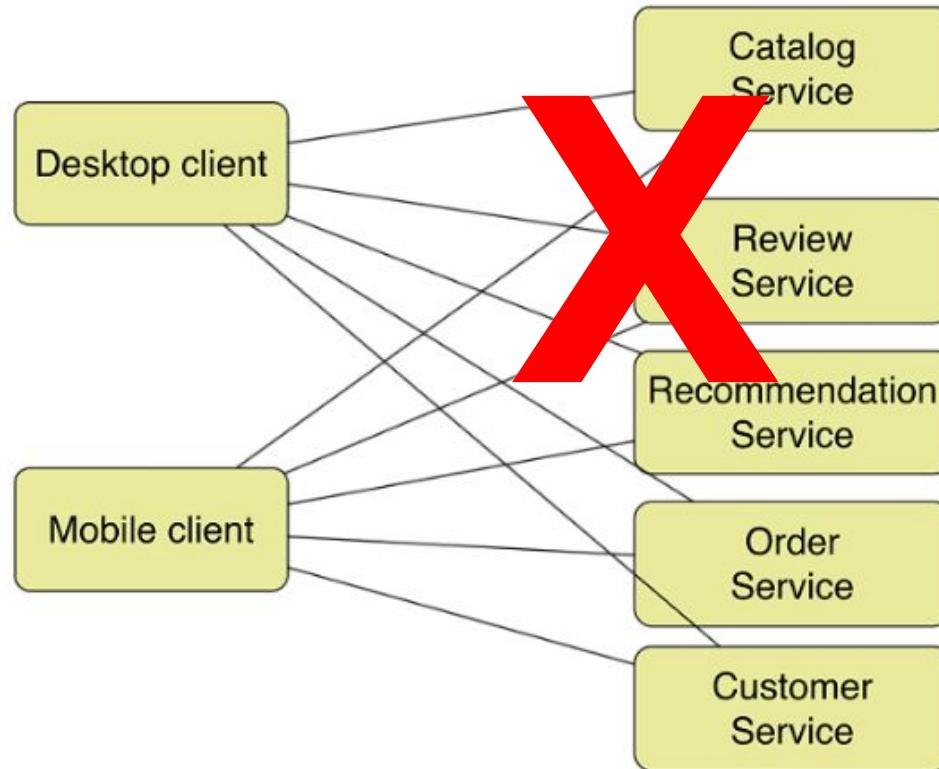


Times atuando na evolução Microsserviço



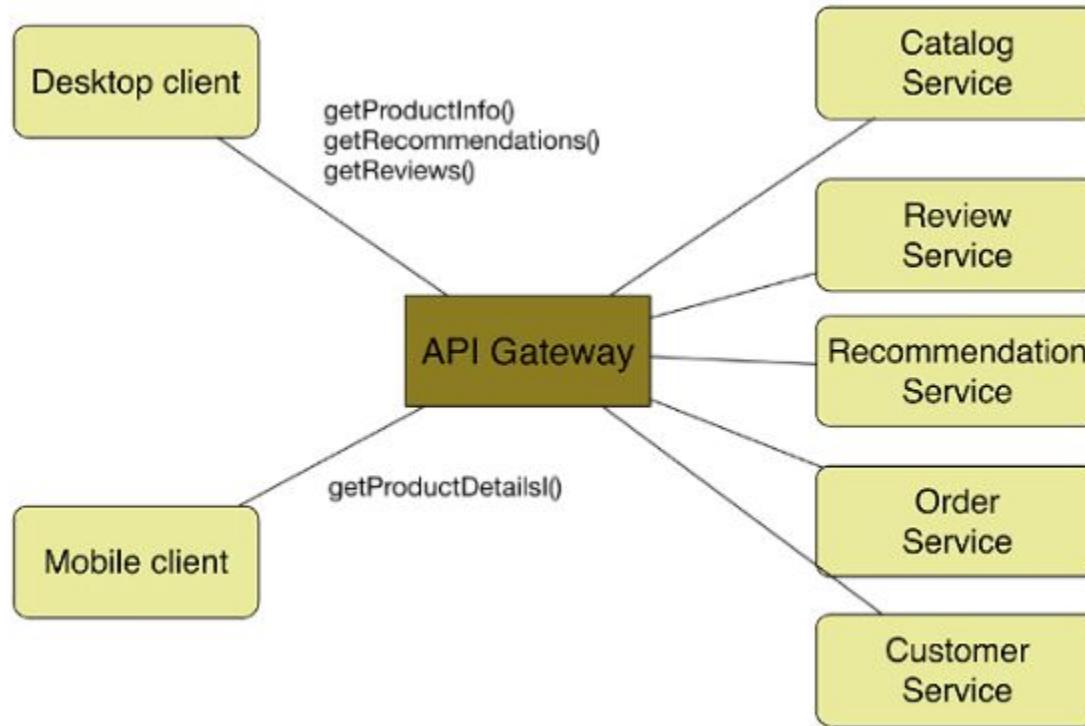


Canal comunicando com Microsserviço



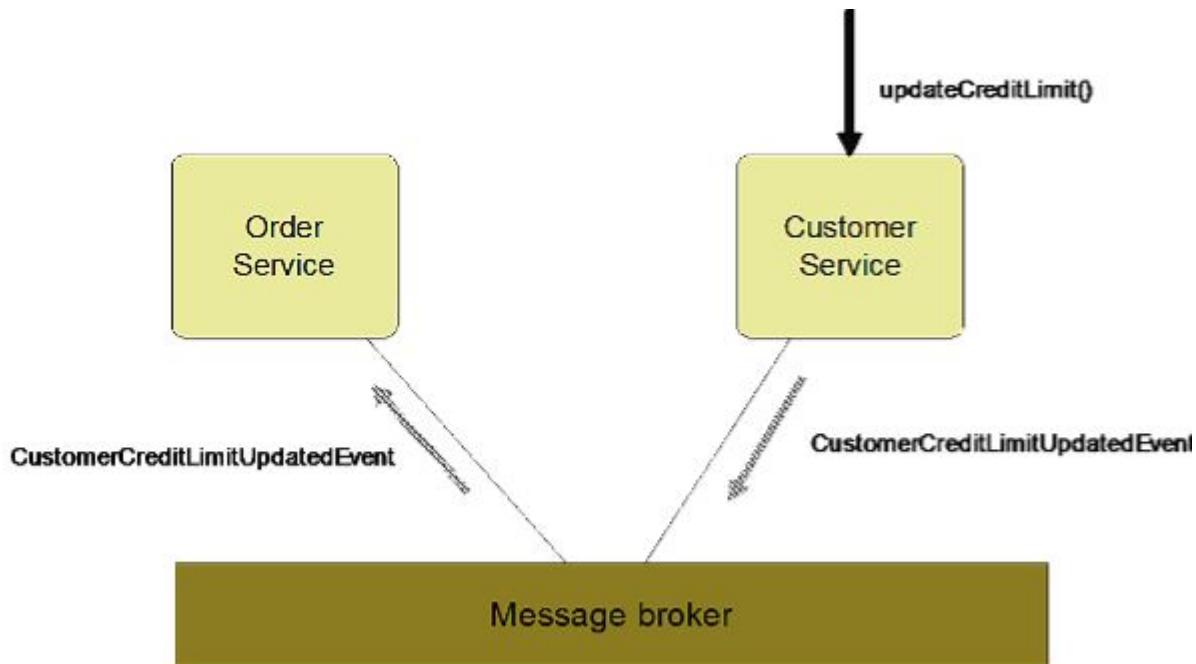


Canal comunicando com Microsserviço com a camada de APIGateway



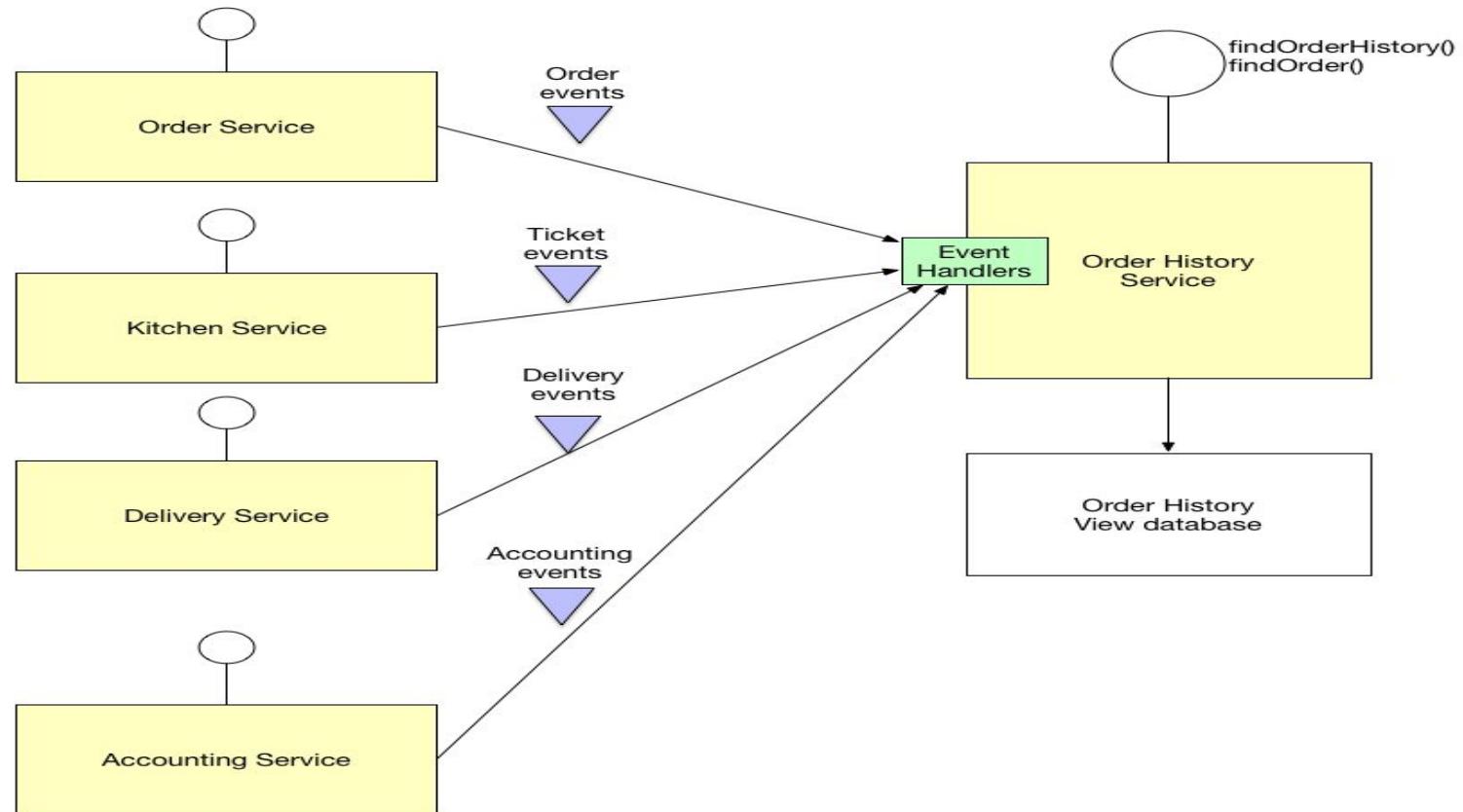


Comunicação interna entre os microsserviços



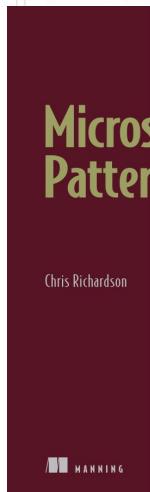


Comunicação interna entre os microsserviços - Command Query Responsibility Segregation (CQRS)





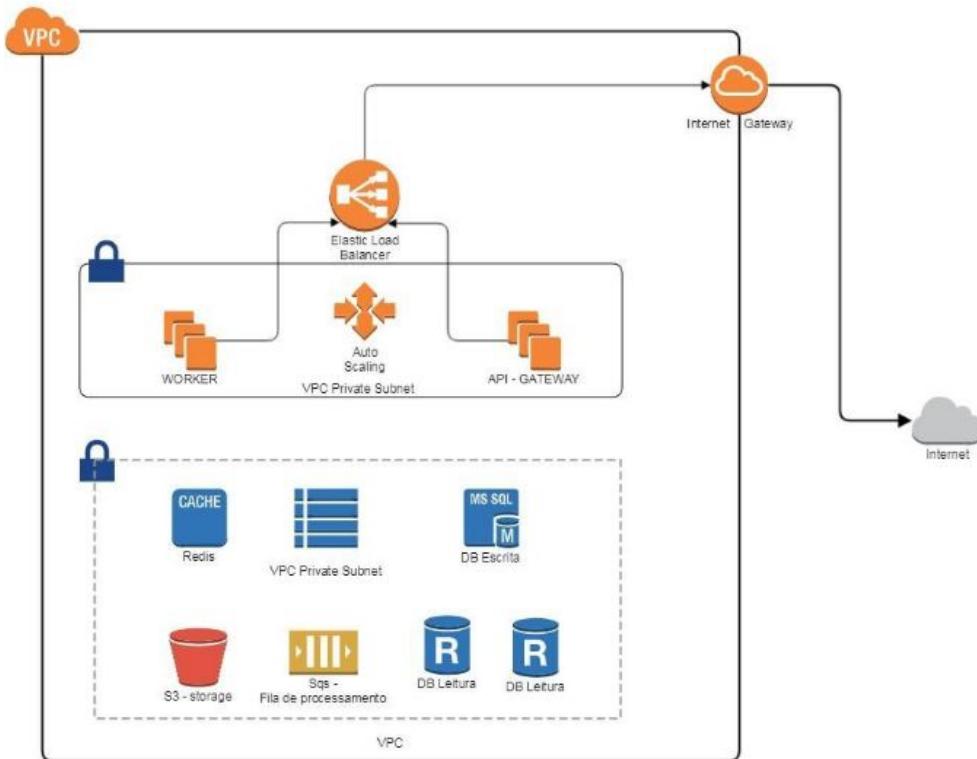
Comunicação interna entre os microsserviços - Command Query Responsibility Segregation (CQRS)



With examples in Java

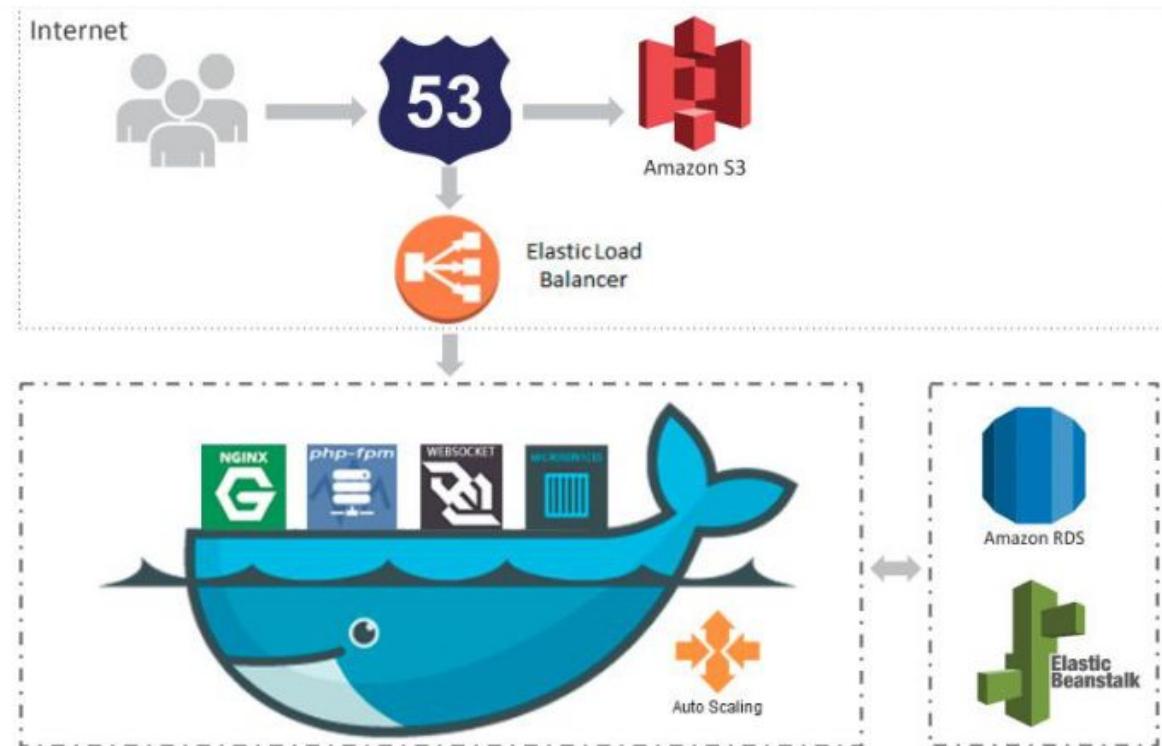


Cloud e microserviços





Cloud e microsserviços



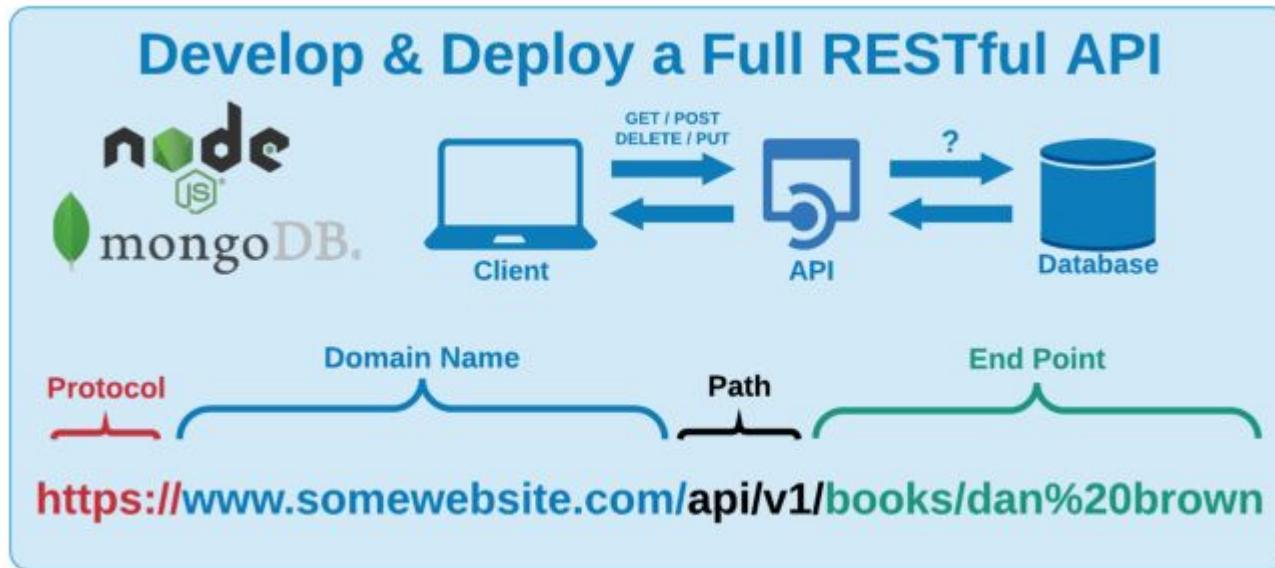


Padrão Restful



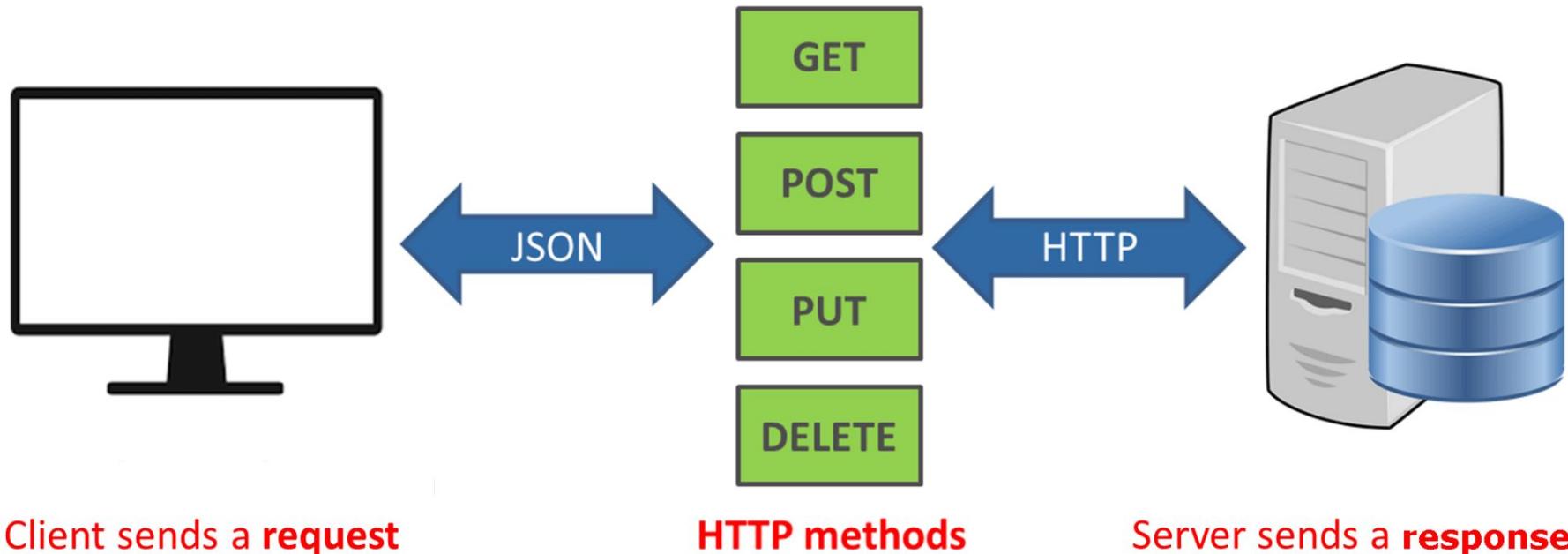


Padrão Restful



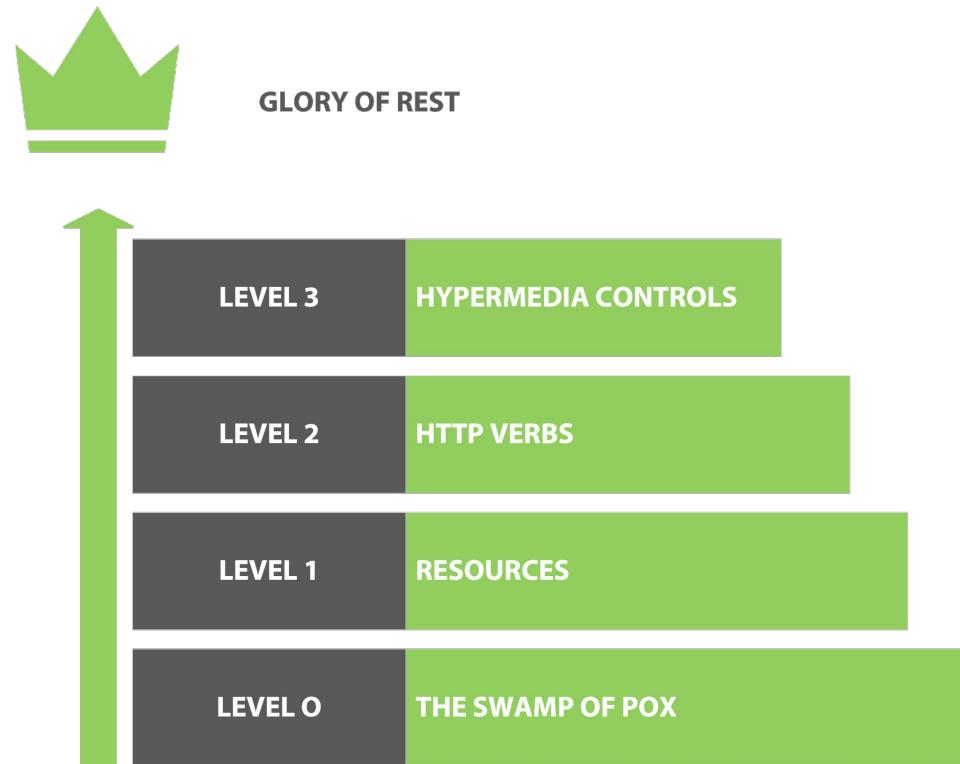


Padrão Restful





Padrão Restful





Padrão Restful

The screenshot shows a web browser displaying the "Newscoop REST API documentation". The URL in the address bar is `newscoop.example.com/documentation/rest-api/`. The page title is "Newscoop REST API documentation". A dropdown menu for "body format" is set to "Form Data". Below the title, there are several API endpoint cards:

- LINK** `/api/articles/{number}/{language}.{_format}` Link resource with Article entity
- UNLINK** `/api/articles/{number}/{language}.{_format}` Unlink resource from Article
- GET** `/api/attachments.{_format}` Get all attachments
- POST** `/api/attachments.{_format}` Create new attachment
- POST | PUT** `/api/attachments/{number}.{_format}` Update attachment
- GET** `/api/attachments/{number}.{_format}` Get attachment
- DELETE** `/api/attachments/{number}.{_format}` Delete image

Spring.IO

spring initializer



Spring initializr

≡  **spring initializr**

Project Maven Project Gradle Project **Language** Java Kotlin Groovy

Spring Boot 2.6.1(SNAPSHOT) 2.6.0 2.5.8 (SNAPSHOT) 2.5.7

Project Metadata

Group	com.example
Artifact	demo
Name	demo
Description	Demo project for Spring Boot
Package name	com.example.demo
Packaging	<input checked="" type="radio"/> Jar <input type="radio"/> War
Java	<input type="radio"/> 17 <input checked="" type="radio"/> 11 <input type="radio"/> 8

Dependencies ADD DEPENDENCIES... CTRL + B

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Rest Repositories WEB
Exposing Spring Data repositories over REST via Spring Data REST.

Spring HATEOAS WEB
Eases the creation of RESTful APIs that follow the HATEOAS principle when working with Spring / Spring MVC.

Jersey WEB
Framework for developing RESTful Web Services in Java that provides support for JAX-RS APIs.

Spring Security SECURITY
Highly customizable authentication and access-control framework for Spring applications.

OAuth2 Client SECURITY
Spring Boot integration for Spring Security's OAuth2/OpenID Connect client features.



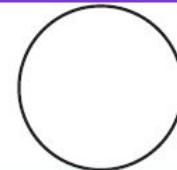
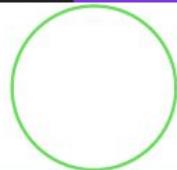
Spring initializr



Testes



Contexto





Redução de custos

Inicialmente, o investimento pode ser maior, mas, após a implementação, o custo do teste automatizado, torna-se menor comparando-se ao teste manual devido à otimização dos procedimentos de teste.



Eficiência nas operações

Ao analisamos a eficiência dos testes manuais, é perceptível que por vezes estes podem levar tempo para serem executados, principalmente em cenários de desenvolvimento muito complexos. Neste caso, o período de execução para o teste automatizado é muito menor e os resultados podem ser averiguados rapidamente trazendo agilidade para as operações de QA.



Aumento da produtividade

Como o teste automatizado é mais eficiente, ele pode trazer retornos mais rápidos para os desenvolvedores corrigirem erros preventivamente. Isso aumenta a [produtividade da equipe](#) devido ao feedback ativo que os testes automatizados proporcionam.



Segurança de dados

É bastante comum que testes automatizados sejam indicados para a realização de testes não-funcionais como segurança e desempenho, por exemplo.

No caso do quesito segurança, a automação realiza verificações constantes para identificar violações e reduzir o risco de possíveis invasões ou outros tipos de ataques. A aplicação adequada deste tipo de verificação é determinante principalmente para sistemas que necessitam evitar que códigos defeituosos atraiam hackers que podem tentar invadir o sistema para coletar informações valiosas.



Tem Mais Benefícios com Testes



Os testes automatizados levam menos tempo para serem executados do que os testes manuais

Os testes manuais podem ser lentos, principalmente quando há inúmeros deploys. Os testadores devem ler o procedimento, entender os cenários e executar uma ação manual, como digitar um comando ou pressionar um botão e ainda, registrar os resultados. Todos esses procedimentos podem ser substituídos por testes automatizados, permitindo que os testes sejam concluídos em um intervalo de tempo ideal.



Os testes automatizados são menos sujeitos a erros do que os testes manuais

Cansaço, estresse, pressões do dia a dia e trabalho repetitivo podem levar a erros humanos. O teste, quando automatizado elimina essa possibilidade trazendo credibilidade as verificações de resultados esperados, pois o robô segue os passos definidos e não pula execuções.



Os testes automatizados podem ser executados sem qualquer interação do usuário

Outra vantagem dos testes automatizados é a possibilidade de programar o disparo automático da execução dos scripts, sem a necessidade de uma ação humana. Permitindo por exemplo um teste diário do conjunto “x” de cenários, e até mesmo testar o comportamento do sistema em horários e dias alternativos, como sábado e domingo.



Os testes automatizados podem ser executados em paralelo

Enquanto um testador humano só pode fazer uma coisa por vez, os robôs de automação podem simular vários testes ao mesmo tempo. Com testes automatizados bem arquitetados podemos verificar diferentes funcionalidades, em diferentes navegadores e sistemas operacionais, todos executados em paralelo e de maneira isolada.



Os testes automatizados podem criar relatórios de teste bem elaborados

Os testes automatizados podem fazer mais do que o próprio teste. Eles também podem automatizar coisas que são normalmente feitos manualmente após a conclusão do teste. Neste caso, a criação de um relatório de teste automatizado que indica tudo que passou, falhou ou não foi executado. Além disso, o teste automatizado pode gerar evidências como prints e vídeos em tempo real de execução.



Os testes são necessários

Dos benefícios mostrados acima, considero dois como os mais importantes: a eficiência e a precisão.

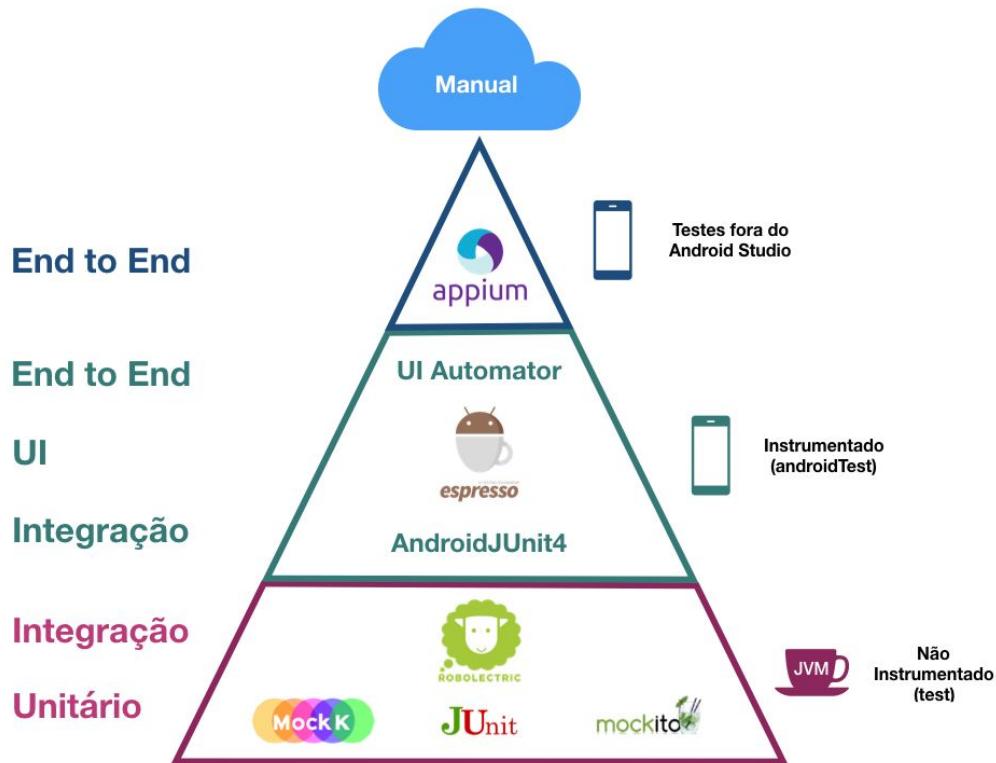
A eficiência dos scripts de testes automatizados são a chave mestre para agregar valor ao processo de teste manual. Ajudam a ganhar vantagem sob prazos curtos, pouca documentação e constantes interações.

Já a precisão, podemos considerar como a joia do processo automatizado, pois a padronização dos códigos proporciona resultados precisos e ajuda bastante para manter a qualidade do software.

Por isso devemos analisar juntamente com a equipe e saber as funcionalidades que devemos investir em automação, para garantirmos entregas mais rápidas e eficientes de qualidade aos clientes finais.



Pirâmide de testes



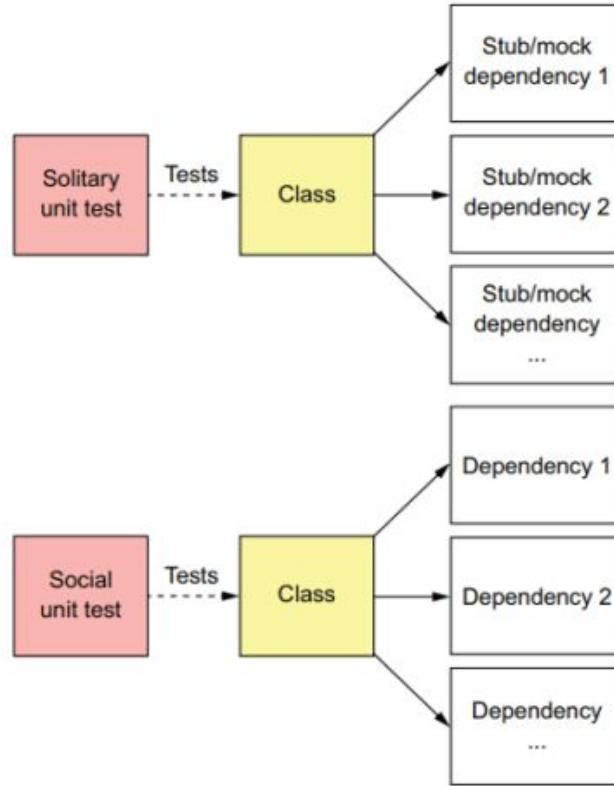
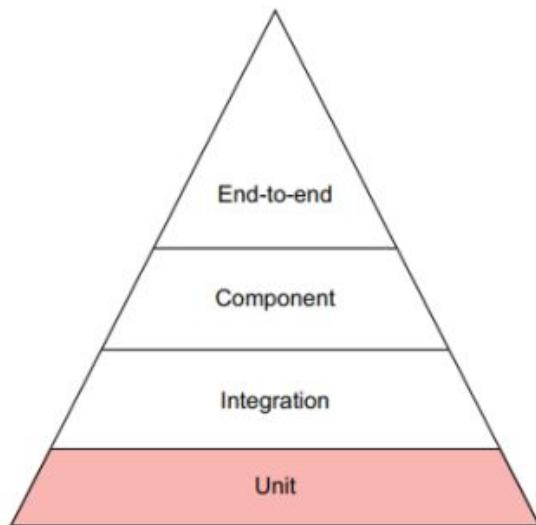


Pirâmide de testes



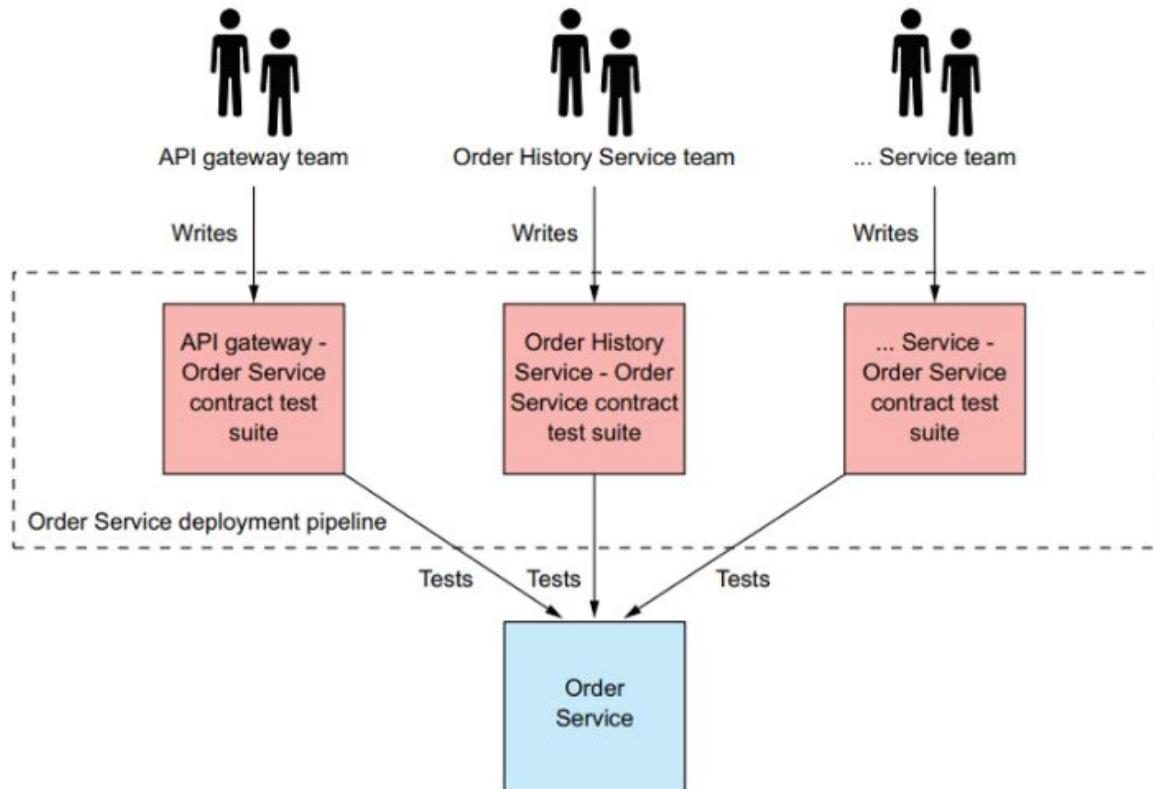


Pirâmide de testes - Unitários



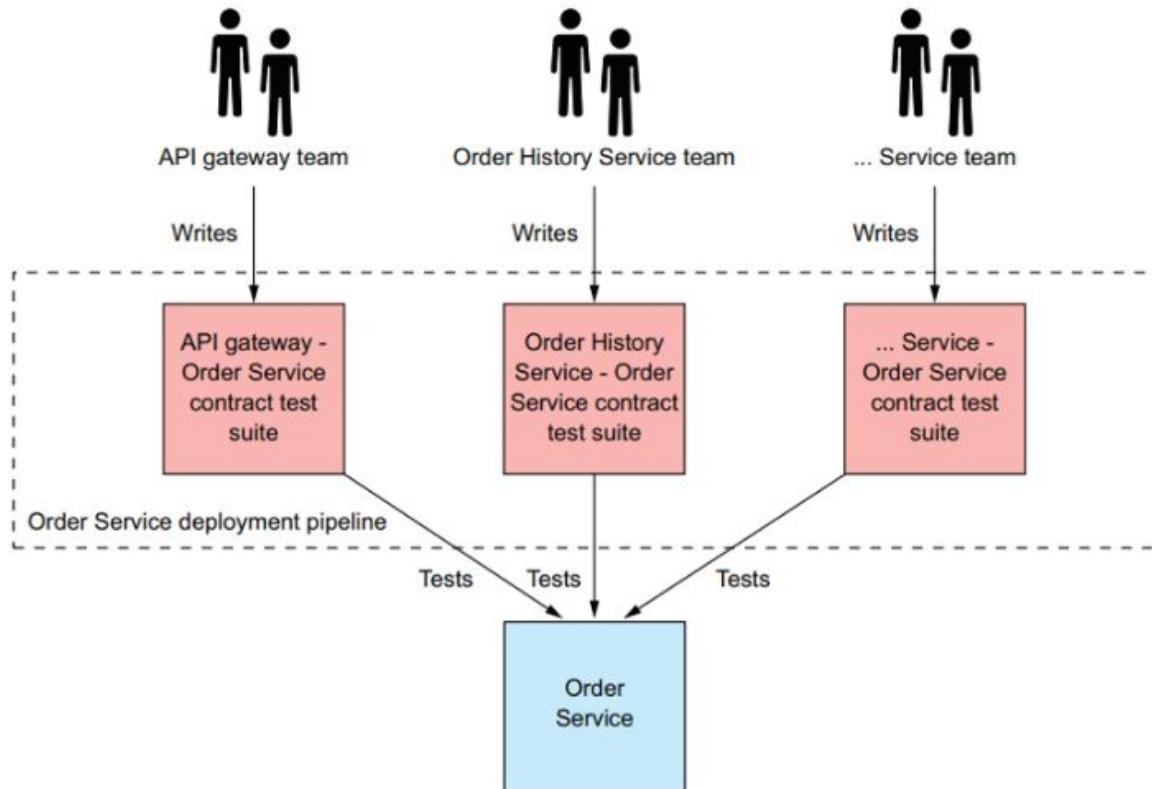


Pirâmide de testes - Integration



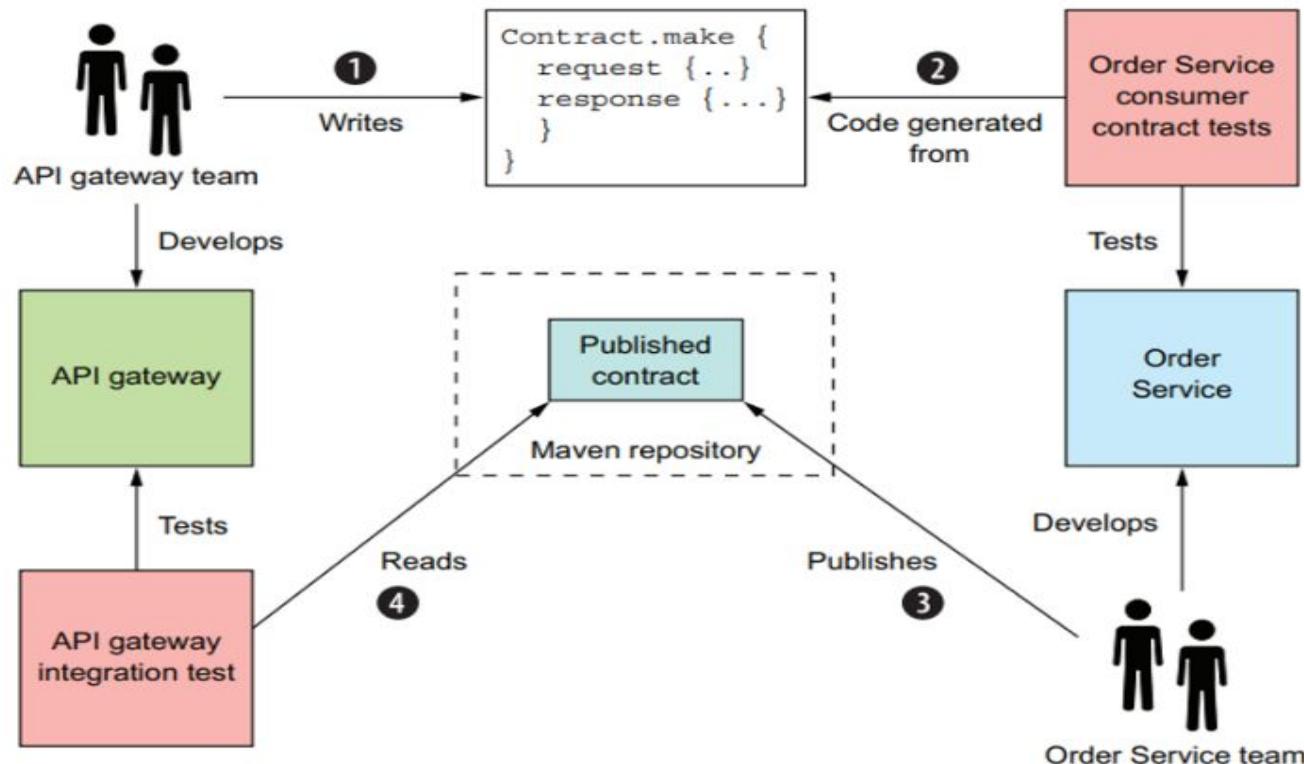


Pirâmide de testes - Integration



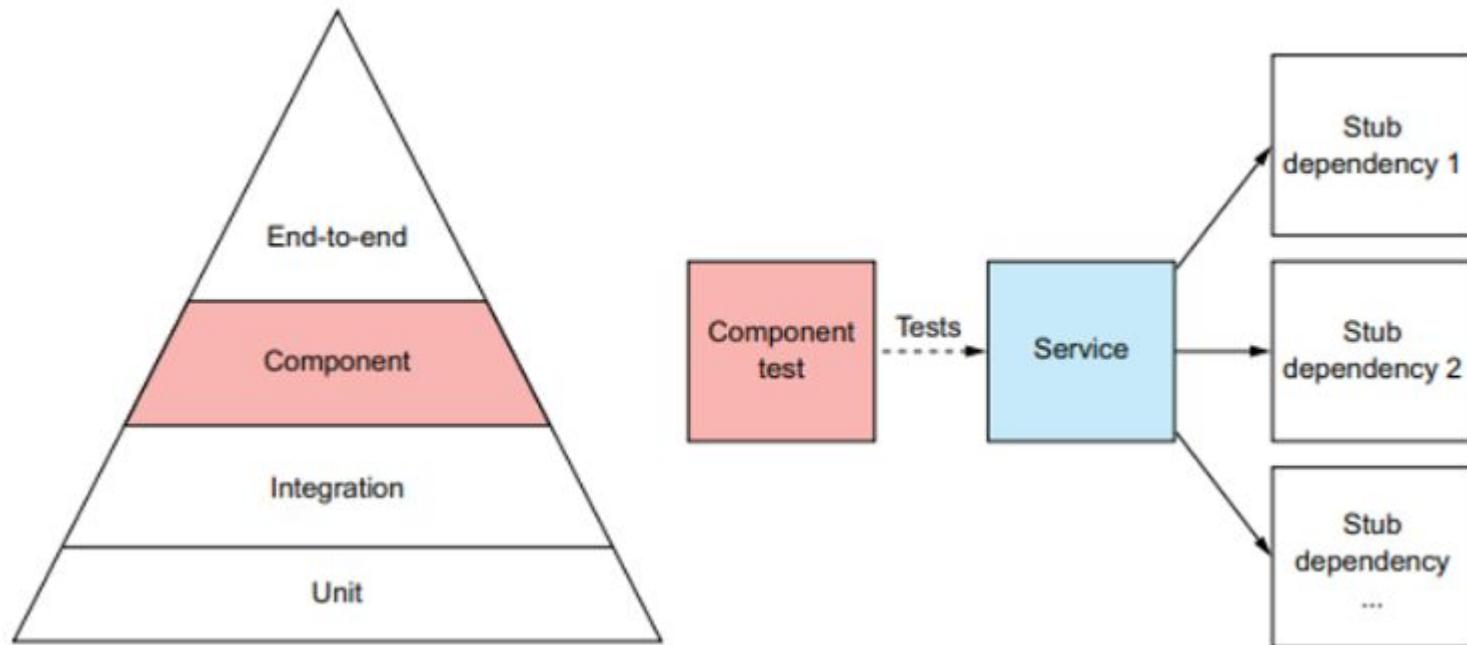


Pirâmide de testes - Integration - Mocks (SpringContract)





Pirâmide de testes - Componente - BDD





Pirâmide de testes - Componente - BDD

Feature: Place Order

As a consumer of the Order Service
I should be able to place an order

Scenario: Order authorized

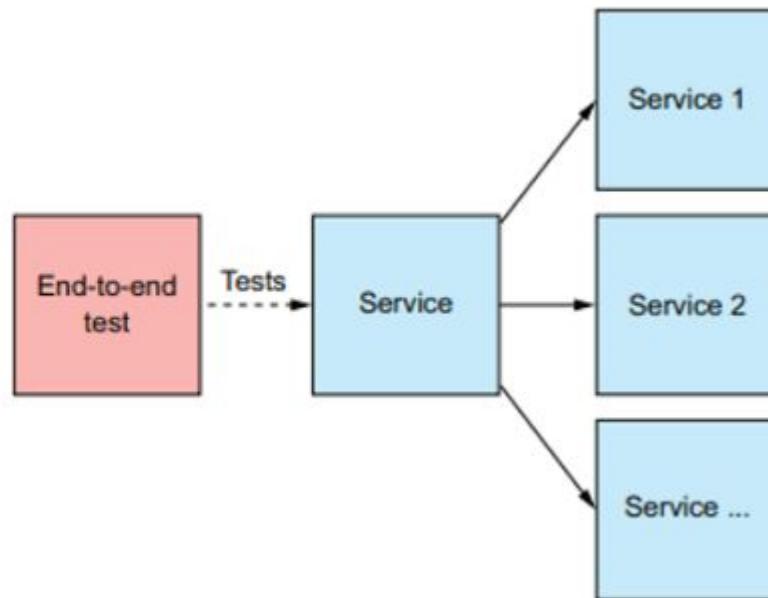
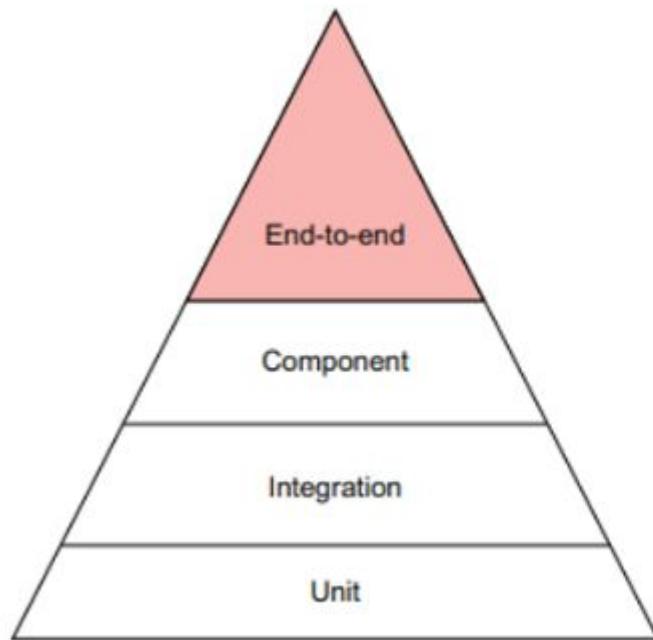
Given a valid consumer
Given using a valid credit card
Given the restaurant is accepting orders
When I place an order for Chicken Vindaloo at Ajanta
Then the order should be APPROVED
And an OrderAuthorized event should be published

Scenario: Order rejected due to expired credit card

Given a valid consumer
Given using an expired credit card
Given the restaurant is accepting orders
When I place an order for Chicken Vindaloo at Ajanta
Then the order should be REJECTED
And an OrderRejected event should be published

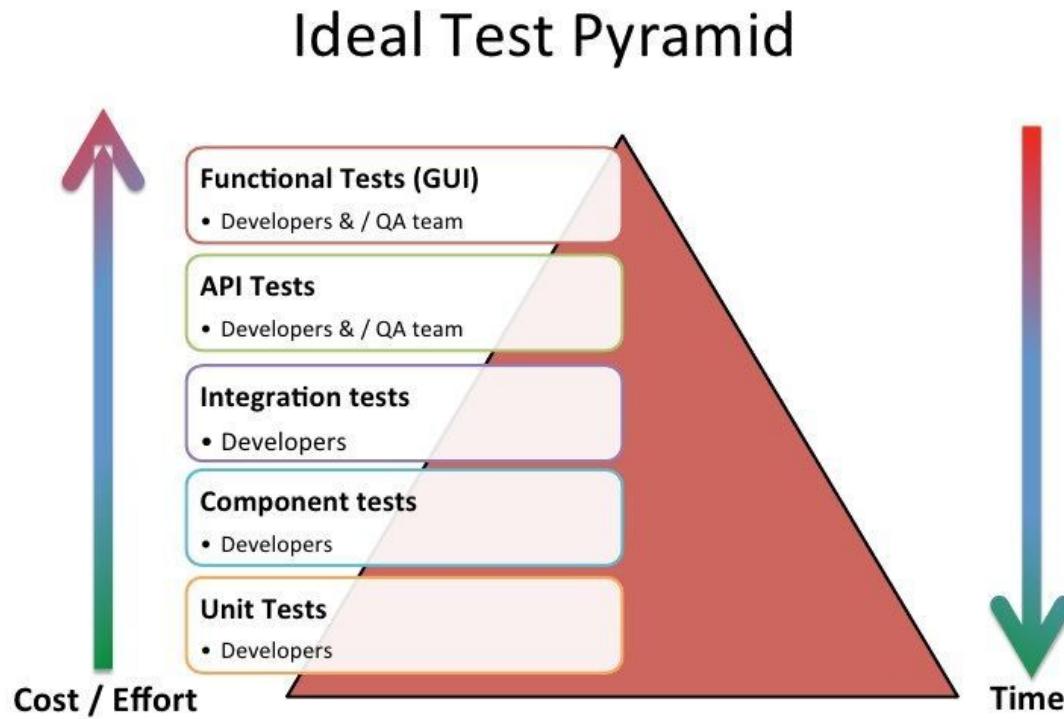


Pirâmide de testes - End-to-end



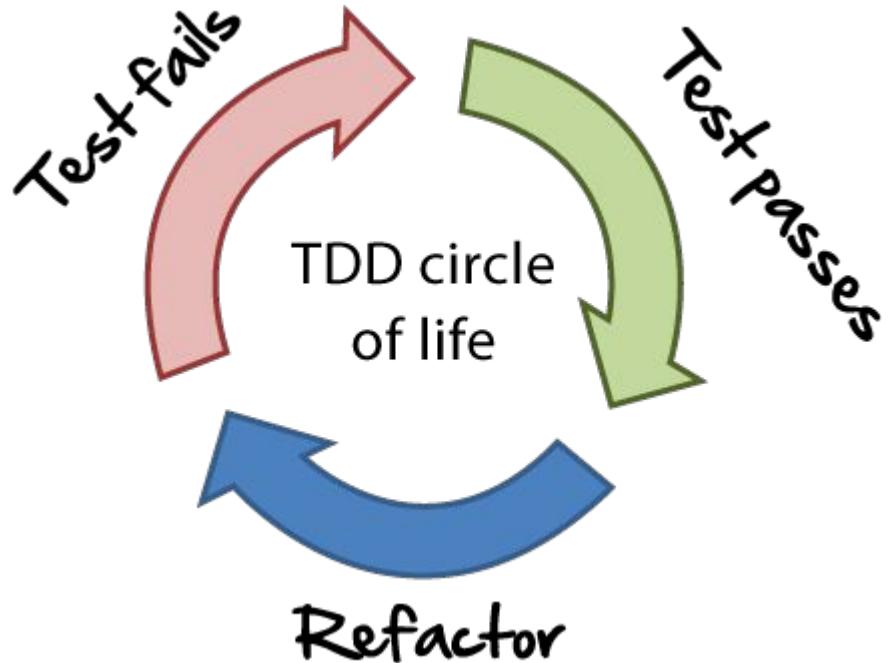


Pirâmide de testes - End-to-end





TDD é a sigla para Test Driven Development



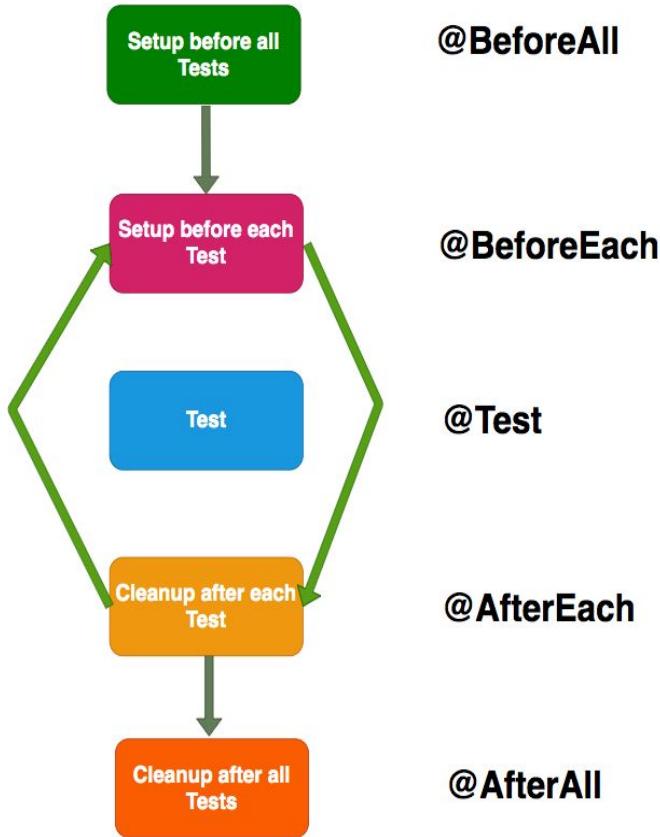


JUnit

JUnit 5

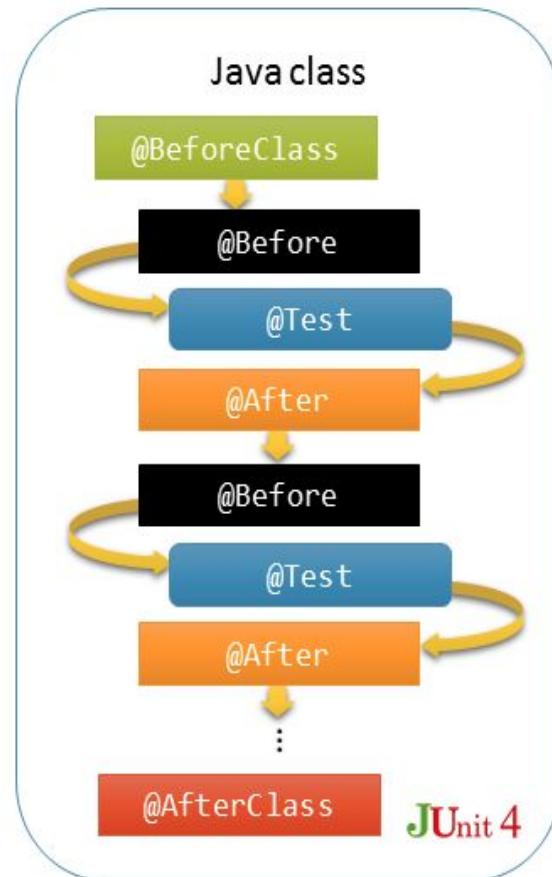


JUnit Ciclo de vida



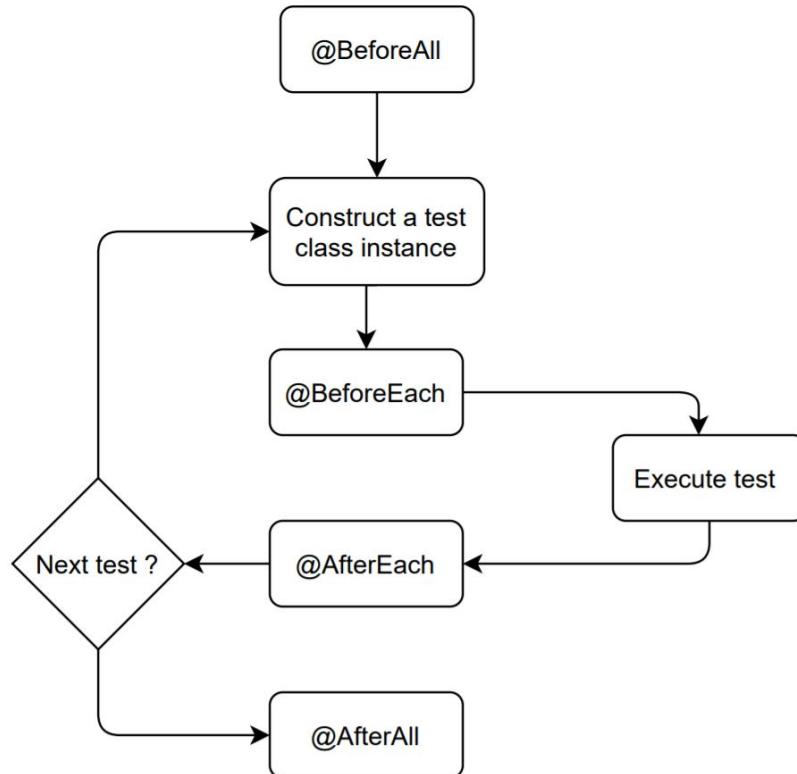


JUnit Ciclo de vida





JUnit Ciclo de vida





JUnit Exemplo

```
import static org.junit.jupiter.api.Assertions.assertEquals;  
  
import example.util.Calculator;  
  
import org.junit.jupiter.api.Test;  
  
class MyFirstJUnitJupiterTests {  
  
    private final Calculator calculator = new Calculator();  
  
    @Test  
    void addition() {  
        assertEquals(2, calculator.add(1, 1));  
    }  
}
```



JUnit Exemplo

A standard test class

```
import static org.junit.jupiter.api.Assertions.fail;
import static org.junit.jupiter.api.Assumptions.assumeTrue;

import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;

class StandardTests {

    @BeforeAll
    static void initAll() {
    }

    @BeforeEach
    void init() {
    }

    @Test
    void succeedingTest() {
    }

    @Test
    void failingTest() {
        fail("a failing test");
    }
}
```

```
@Test
@Disabled("for demonstration purposes")
void skippedTest() {
    // not executed
}

@Test
void abortedTest() {
    assumeTrue("abc".contains("Z"));
    fail("test should have been aborted");
}

@AfterEach
void tearDown() {
}

@AfterAll
static void tearDownAll() {
}

}
```



JUnit Exemplo

```
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

@DisplayName("A special test case")
class DisplayNameDemo {

    @Test
    @DisplayName("Custom test name containing spaces")
    void testWithDisplayNameContainingSpaces() {
    }

    @Test
    @DisplayName("J°□°) J")
    void testWithDisplayNameContainingSpecialCharacters() {
    }

    @Test
    @DisplayName("😱")
    void testWithDisplayNameContainingEmoji() {
    }

}
```



JUnit Exemplo componente

```
@Test
public void givenGreetURI_whenMockMVC_thenVerifyResponse() {
    MvcResult mvcResult = this.mockMvc.perform(get("/greet"))
        .andDo(print()).andExpect(status().isOk())
        .andExpect(jsonPath("$.message").value("Hello World!!!"))
        .andReturn();

    Assert.assertEquals("application/json; charset=UTF-8",
        mvcResult.getResponse().getContentType());
}
```

Docker



mini ambiente





O que são Docker Containers?

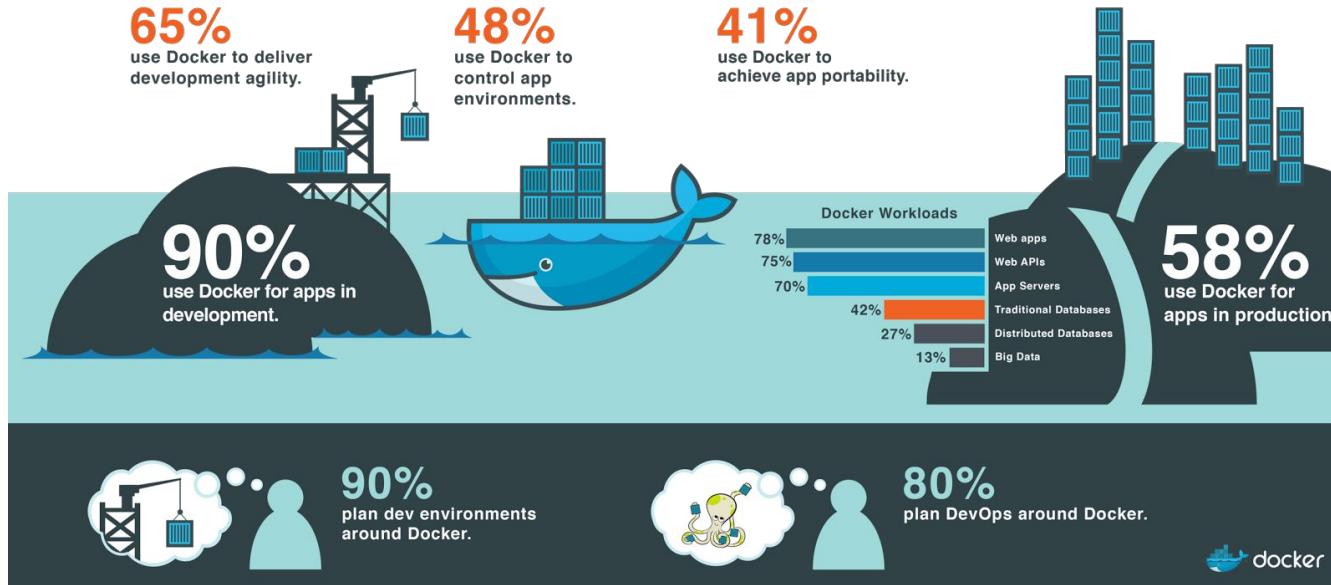
Simplificando, um **contêiner** é um **ambiente isolado** dentro de um **servidor**. Como exemplo: imagine um trem de carga com diversos containers de mercadorias. Se em um dos containers a mercadoria estragar, isso não vai afetar os outros containers, pois cada um deles está isolado e protegido.





O que são Docker Containers?

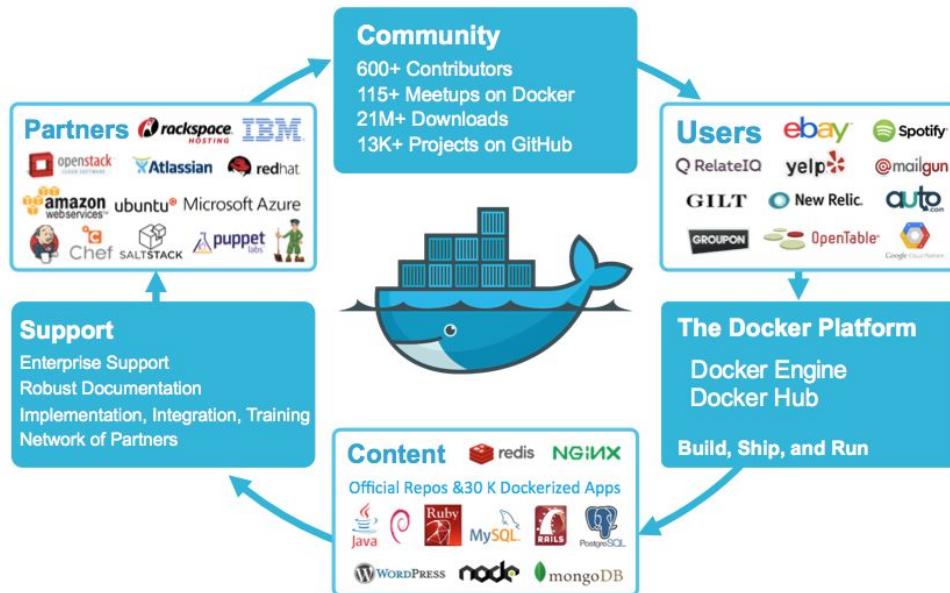
Docker é uma tecnologia de virtualização que possibilita o empacotamento de uma aplicação ou ambiente inteiro dentro de um contêiner.





O que são Docker Containers?

O Docker foi criado com o objetivo de **facilitar o desenvolvimento, acelerar a implantação e a execução** de aplicações em ambientes isolados. Ele foi desenhado especialmente para disponibilizar aplicações da forma mais rápida possível, apoiado pelo modelo DevOps.

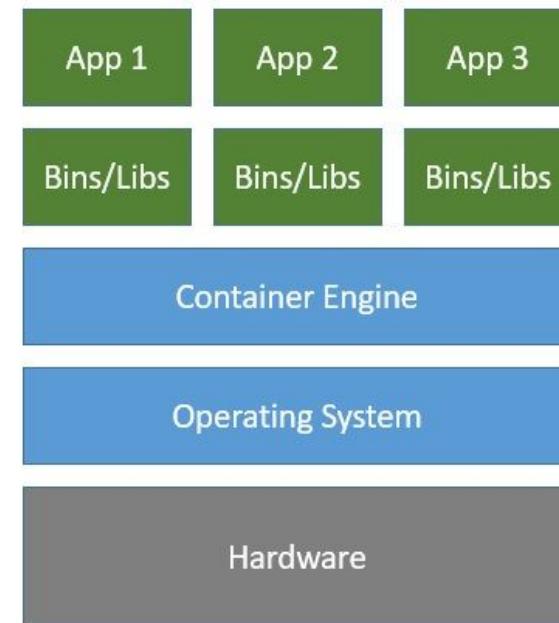




Virtual Machine Vs Docker



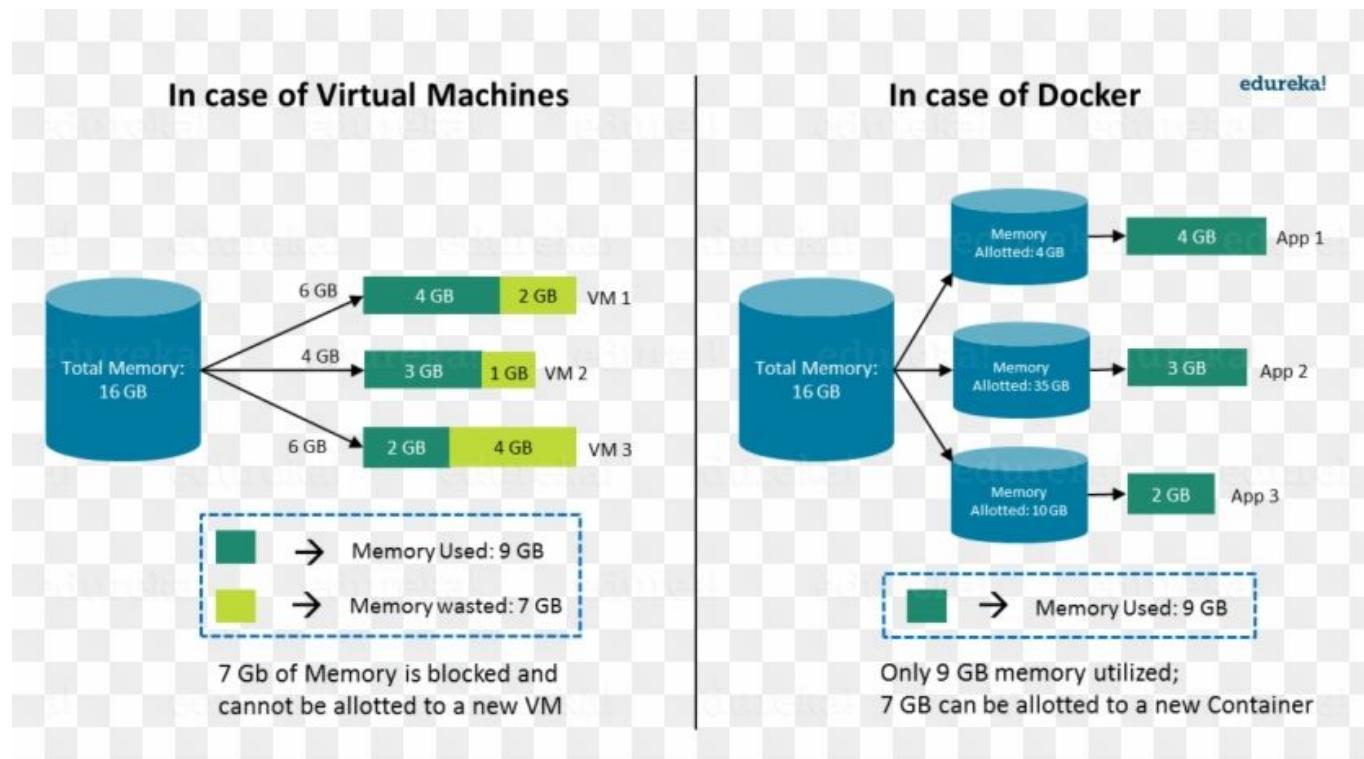
Virtual Machines



Containers

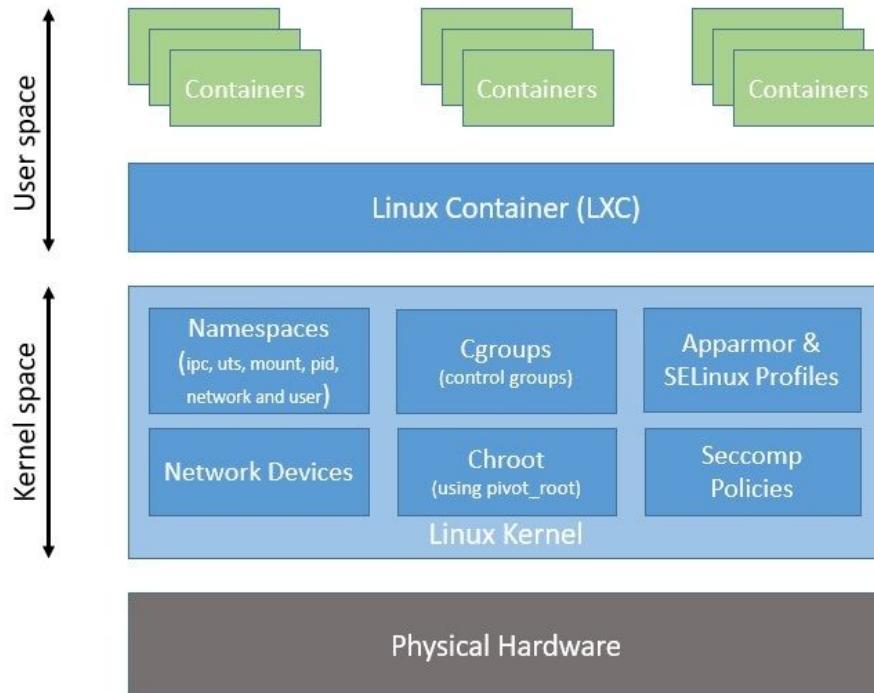


O que são Docker Containers?



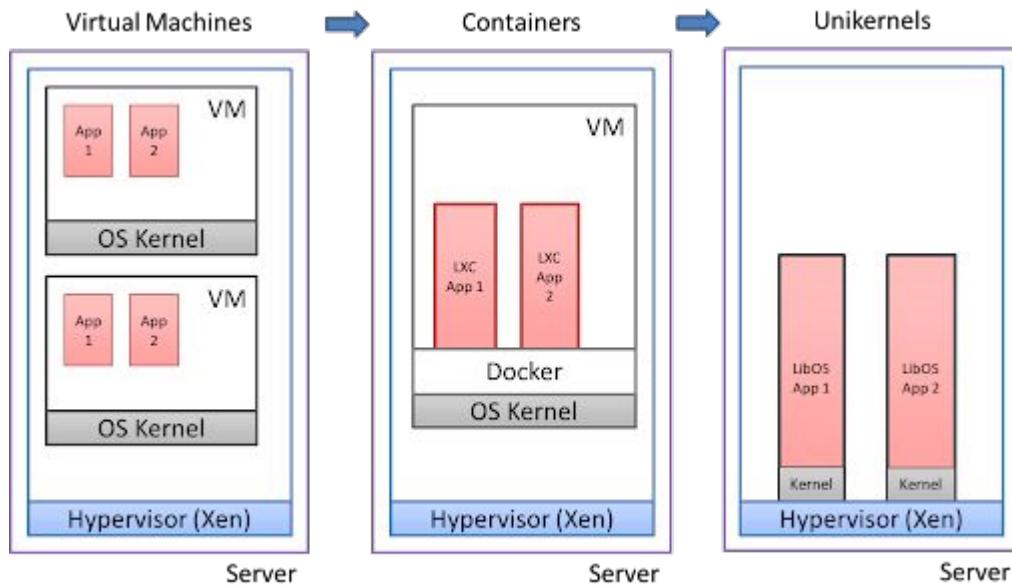


Camadas Docker



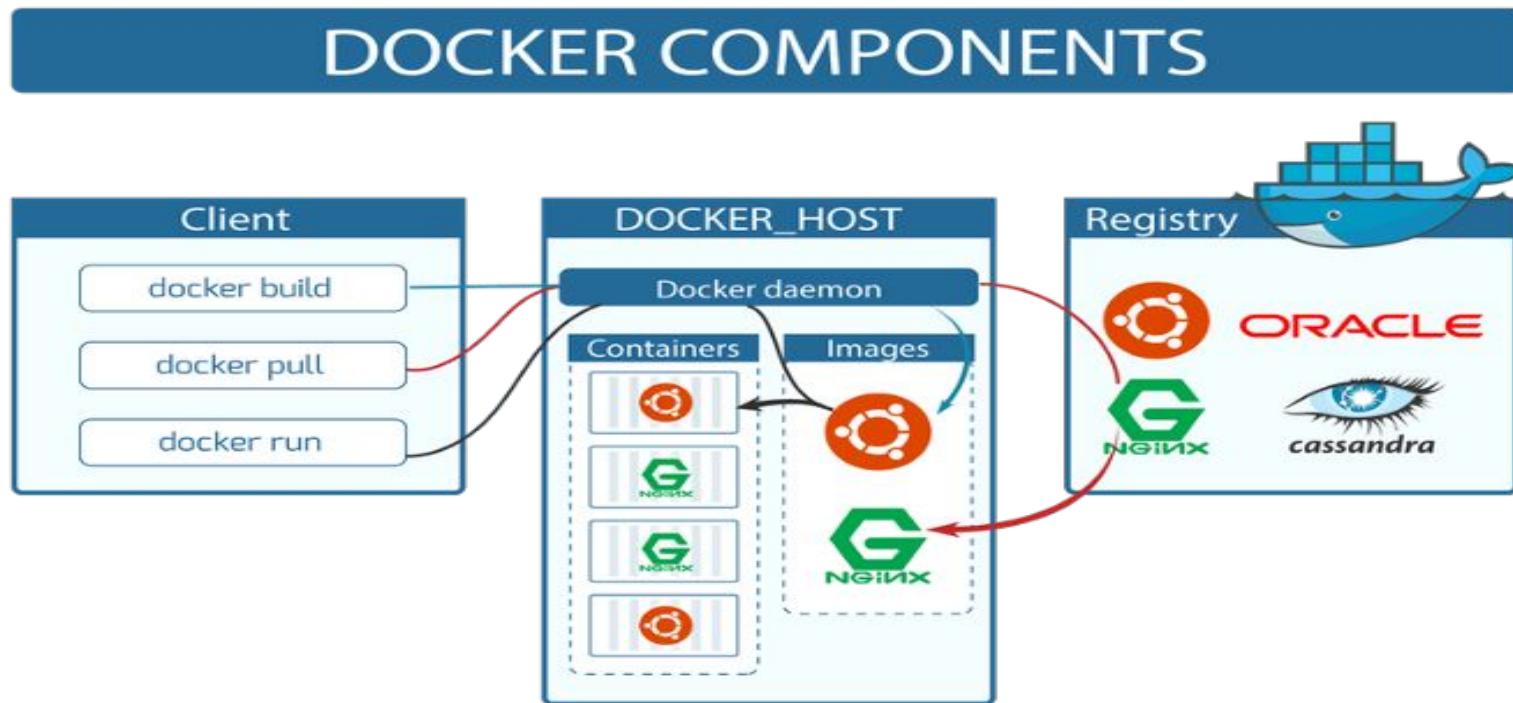


Mais exemplos





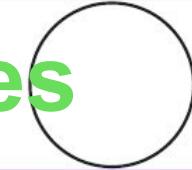
Componentes docker



Mensagerias



Desacoplando aplicações





Integrações entre sistemas

O que é integração de sistemas?

É comum que uma empresa disponha de um conjunto robusto de ferramentas digitais com propósitos distintos entre si, por exemplo, um [sistema de estoque](#), outro financeiro, um terceiro de logística e por aí vai.

O problema de atuar segundo esse modelo é que muitas informações podem acabar tendo de ser replicadas em cada um deles, o que se traduz em mais trabalho, perda de eficácia e ocorrência de erros e dados duplicados.

O ideal, então, é que cada um dos softwares possa se comunicar com o outro e pedir as informações necessárias para as suas rotinas, melhorando a agilidade e o fluxo de dados dentro da empresa. Isso é a integração dos sistemas.



Integrações entre sistemas

Tipos de integração de sistemas

Por Arquivos

Por Banco de dados

Por Componentes distribuídos EJB, CORBA etc

Por APIs SOAP

Por APIs grpc

Por APIs Rest

Por Mensageria (AMQP)



Integrações entre sistemas

Por Arquivos





Integrações entre sistemas

Por Banco de dados

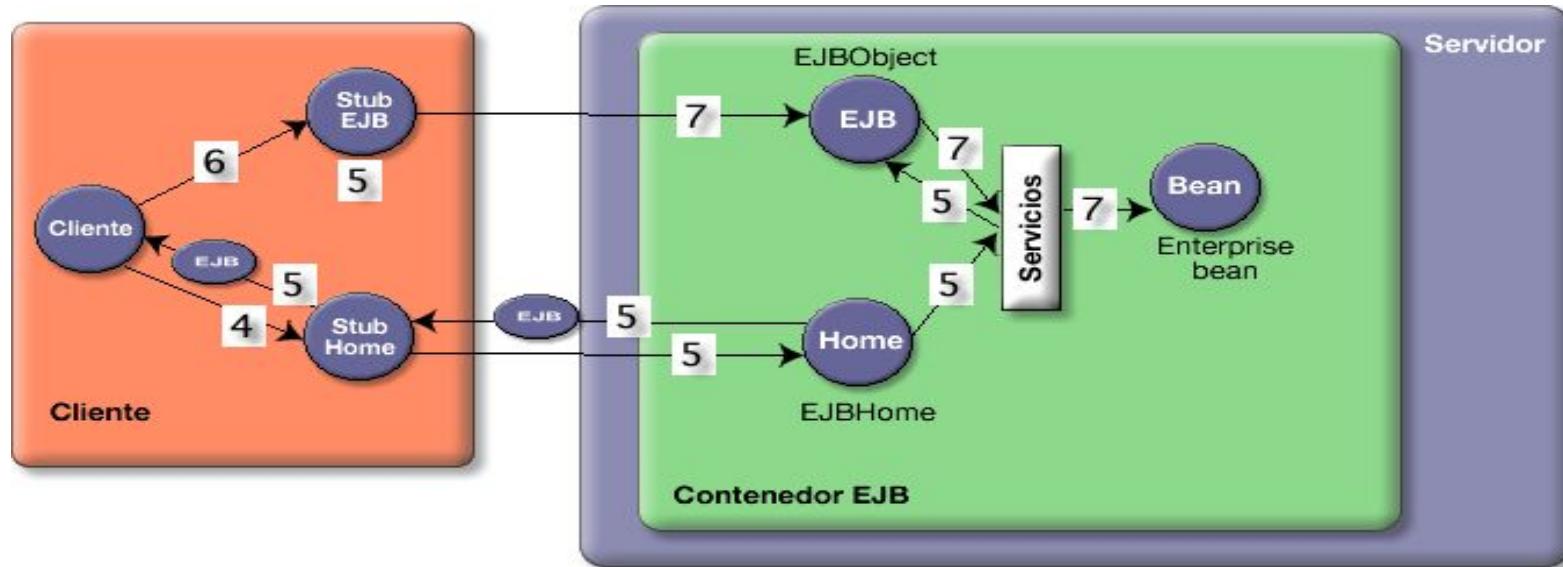




Integrações entre sistemas

Tipos de integração de sistemas

Por Componentes distribuídos EJB, CORBA etc





Integrações entre sistemas

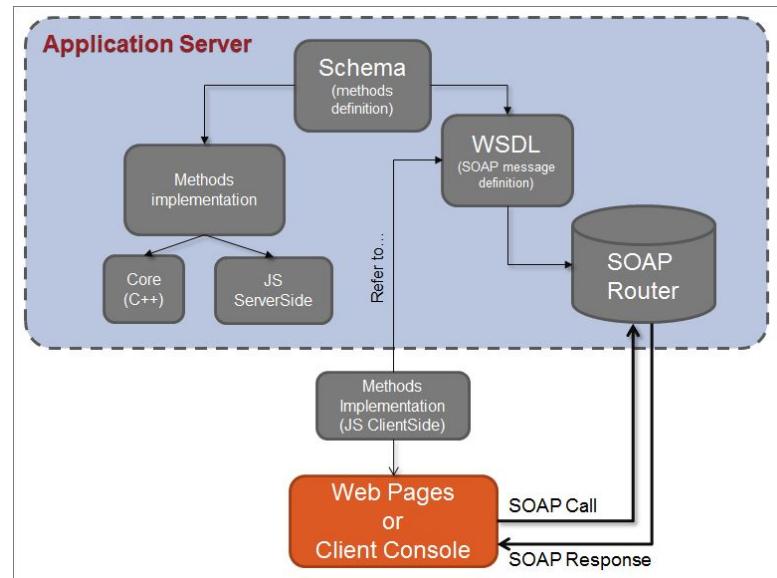
Por APIs SOAP

SOAP WEBSERVICE DIAGRAM

SOAP Request Message through HTTP



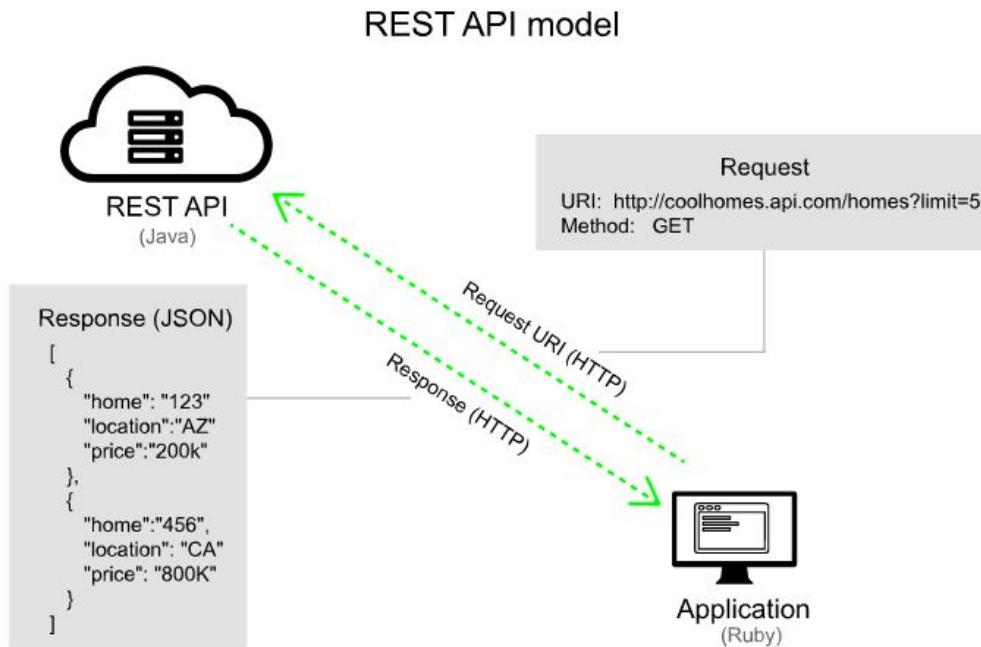
SOAP Response Message through HTTP





Integrações entre sistemas

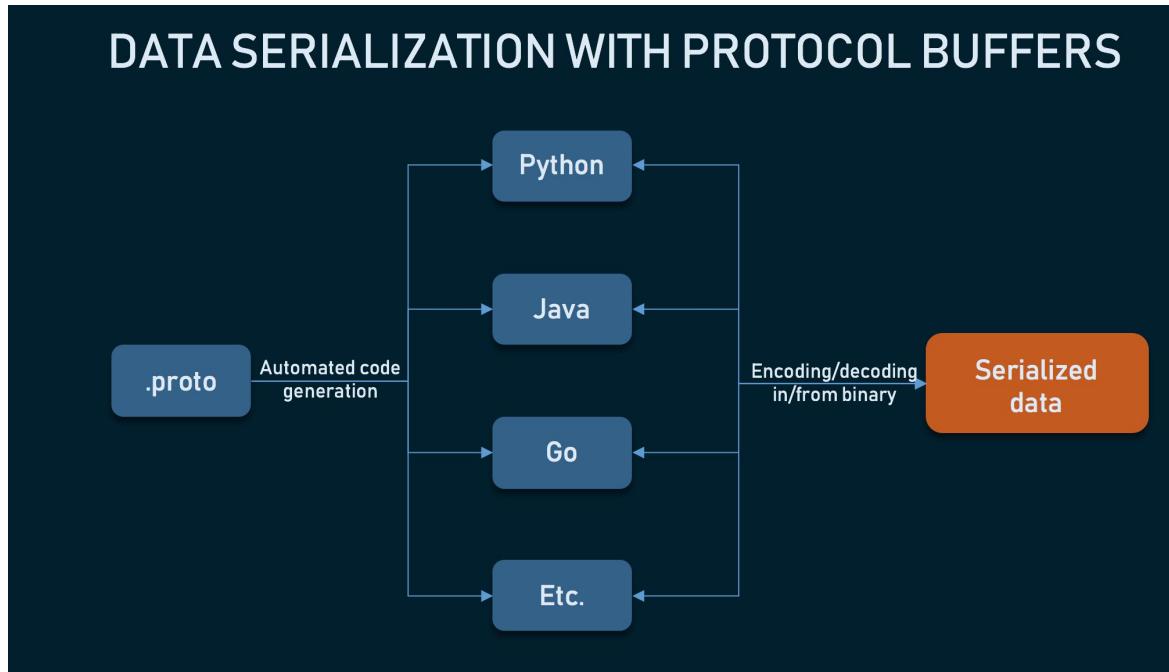
Por APIs Rest





Integrações entre sistemas

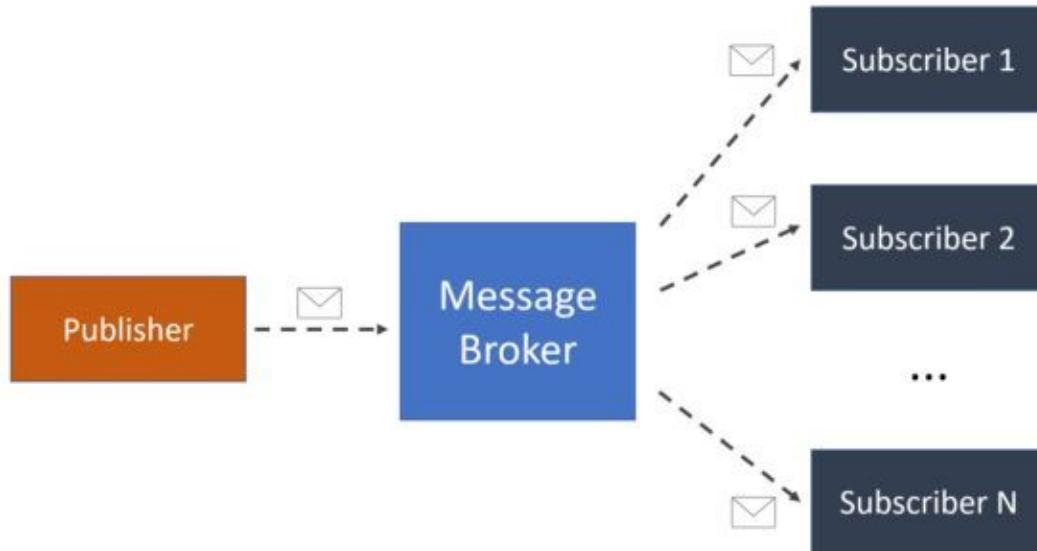
Por APIs grpc





Integrações entre sistemas

Por Mensageria (RabbitMQ/Kafka)





Diferença entre Kafka e RabbitMQ

Tool	Apache Kafka	RabbitMQ
Message Ordering	Provides message ordering because of its partitions. Messages are sent to topics by message key.	Not supported.
Message Lifetime	Kafka is a log, which means that messages are always there. You can manage this by specifying a message retention policy.	RabbitMQ is a queue, so messages are done away with once consumed, and acknowledgment is provided.
Delivery Guarantees	Retains order only inside a partition. In a partition, Kafka guarantees that the whole batch of messages either fails or passes.	Doesn't guarantee atomicity, even in relation to transactions involving a single queue.
Message Priorities	N/A	In RabbitMQ, you can specify message priorities, and consumed message with high priority at the onset.



RabbitMQ principais componentes

Publisher

Consumer

Exchanges

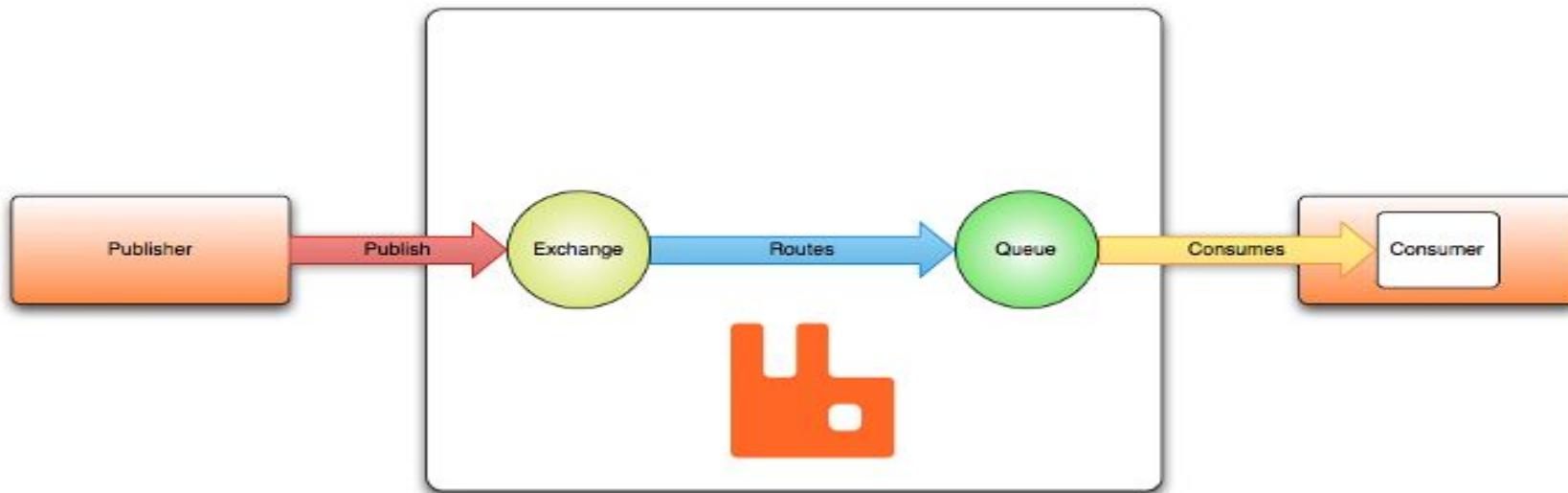
Routing

Queues



RabbitMQ principais componentes

"Hello, world" example routing





RabbitMQ principais componentes

Routing Pattern

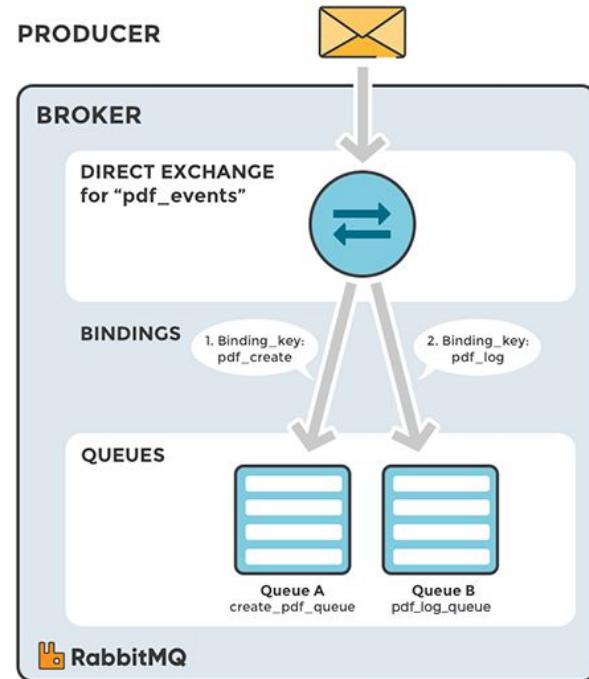
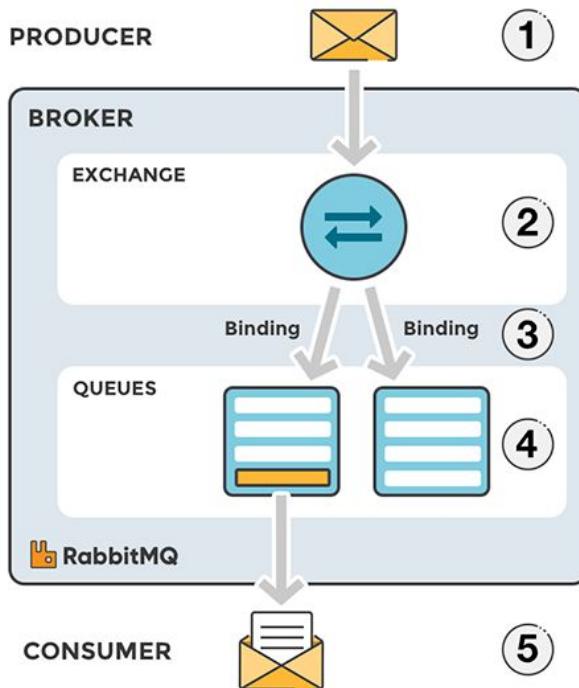
- order.logs.customer
- order.logs.international
- order.logs.personnel
- order.logs.customer.electronics
- order.logs.personnel.electronics
- order.logs.international.electronics

Send Routing Pattern

order.*.*.electronics
order.logs.customer.#

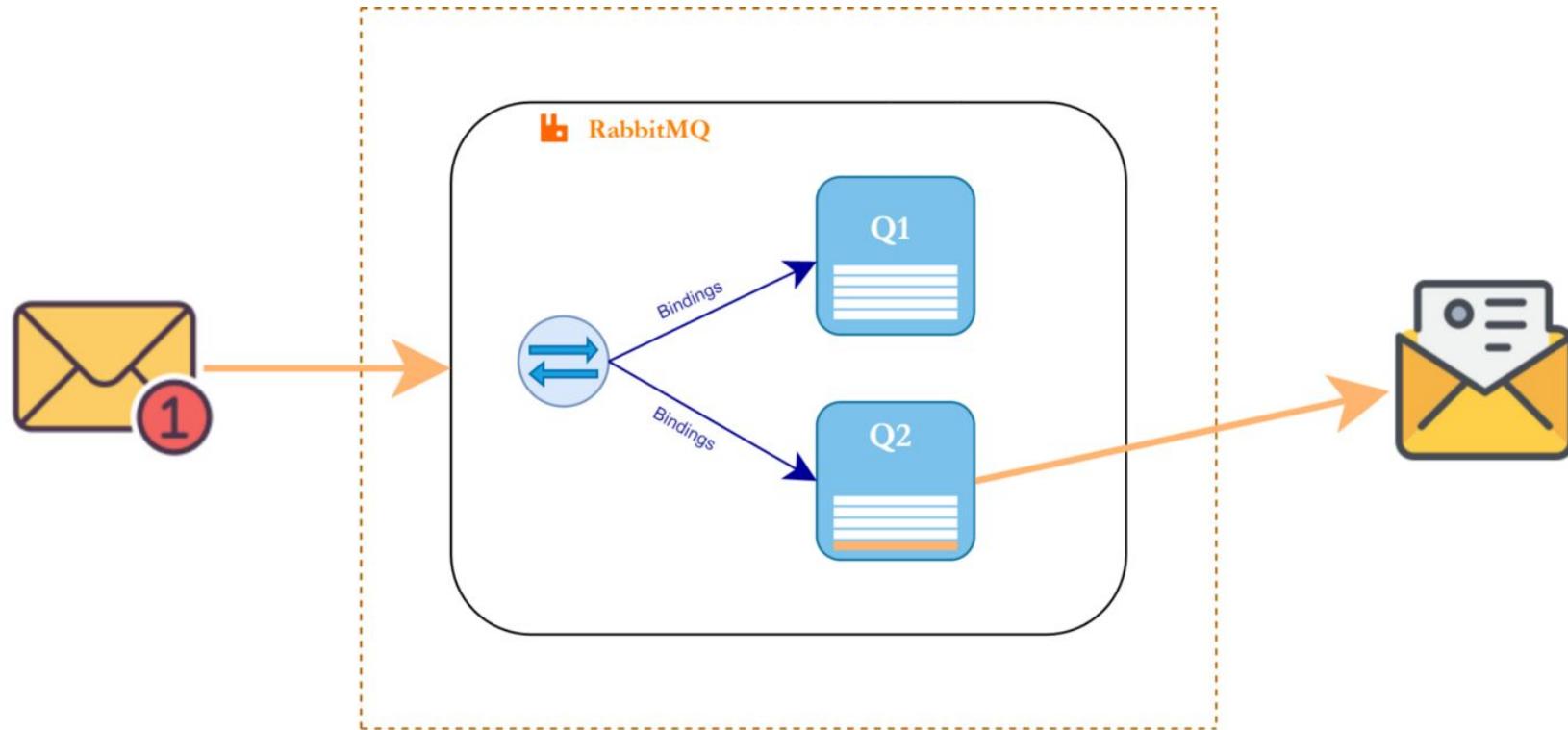


RabbitMQ exchange DIRECT



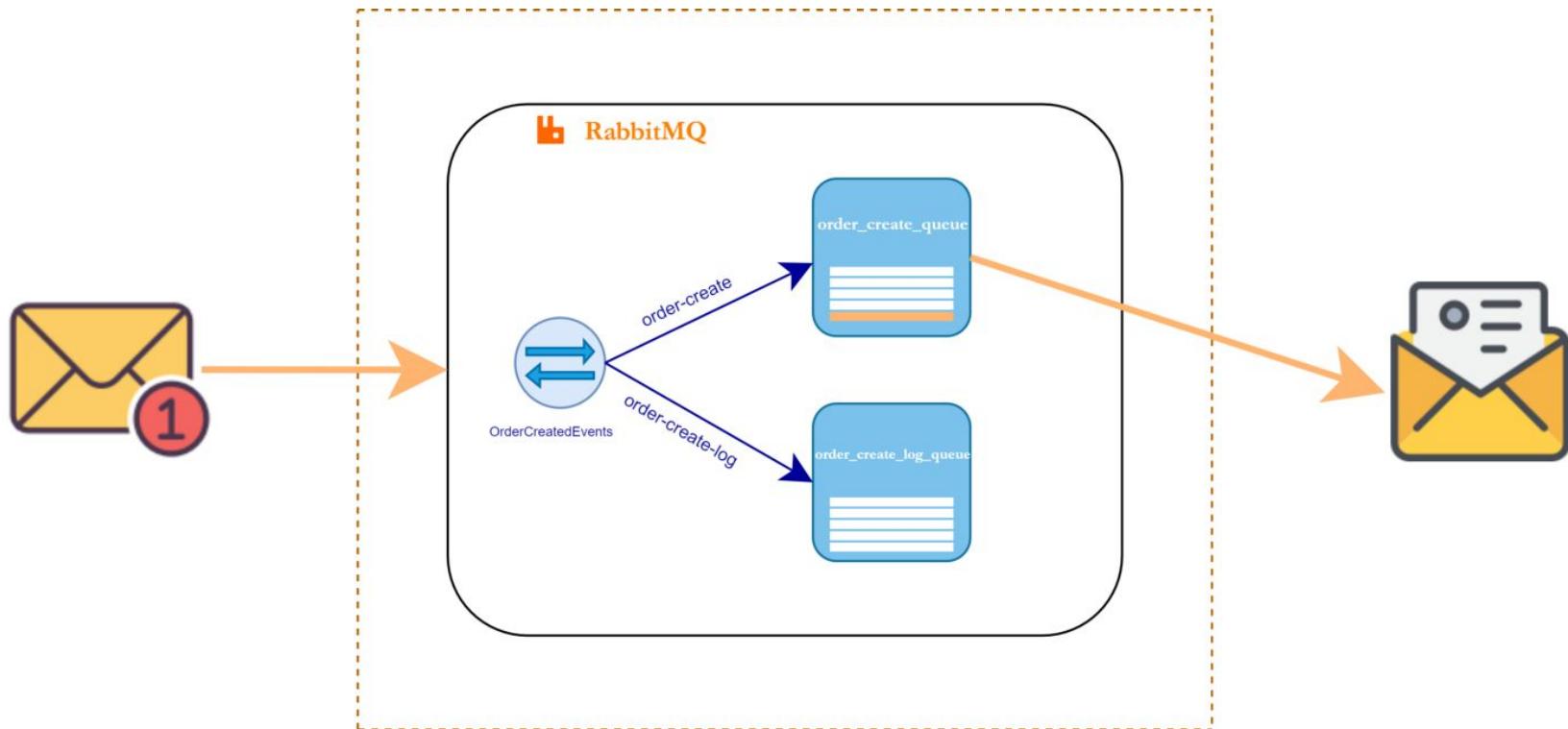


RabbitMQ exchange DIRECT



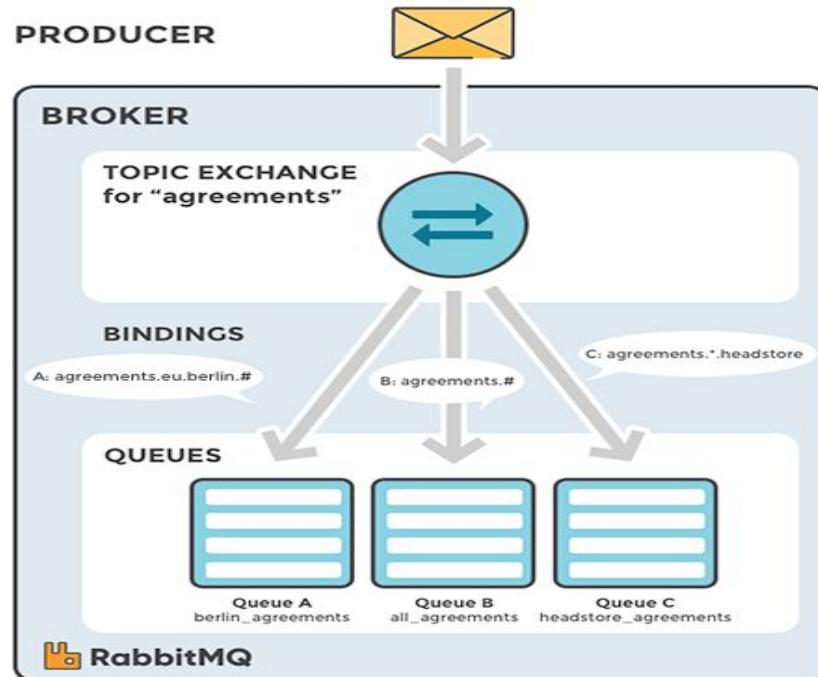


RabbitMQ exchange DIRECT



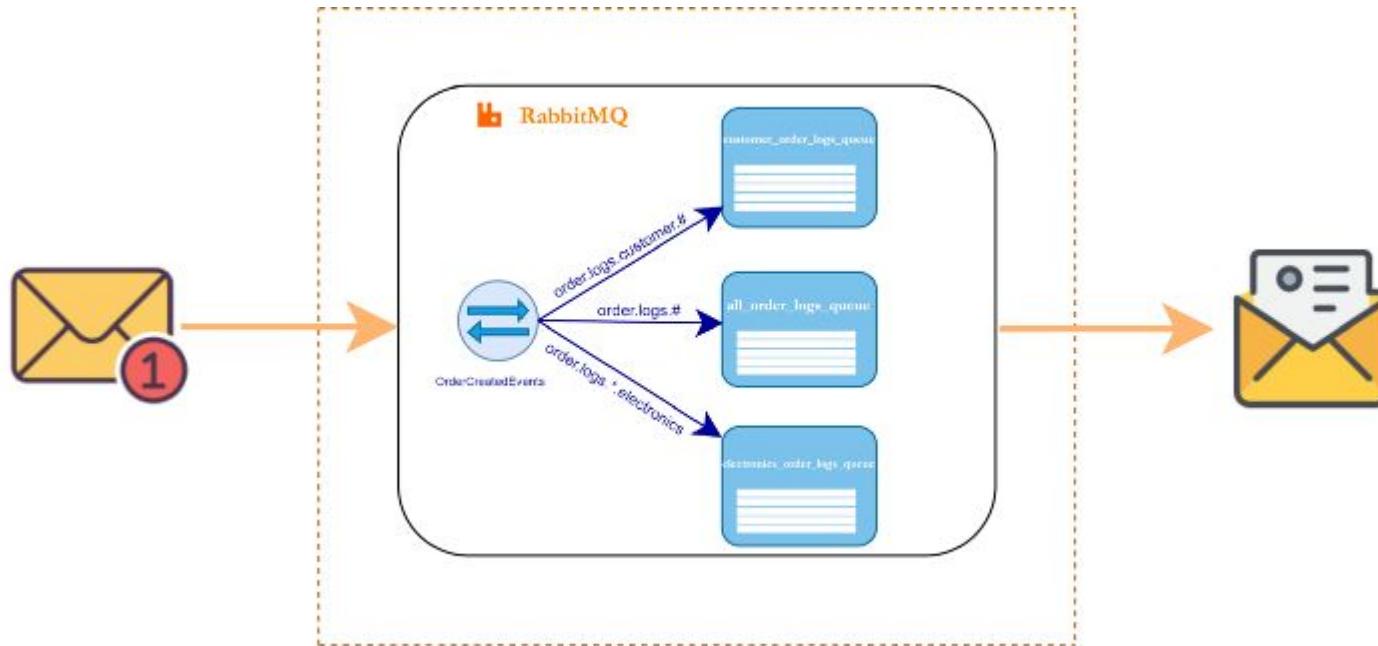


RabbitMQ exchange TOPIC



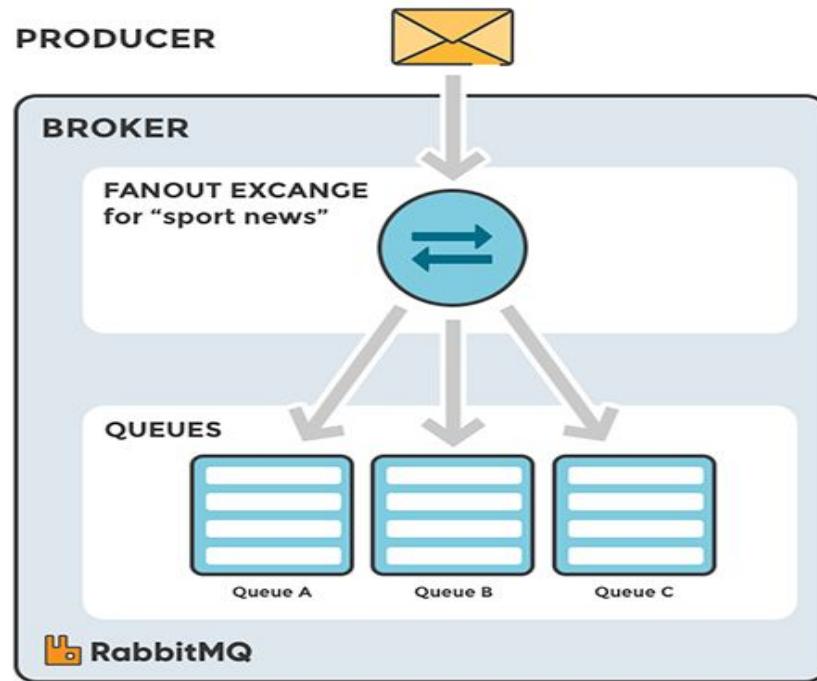


RabbitMQ exchange TOPIC



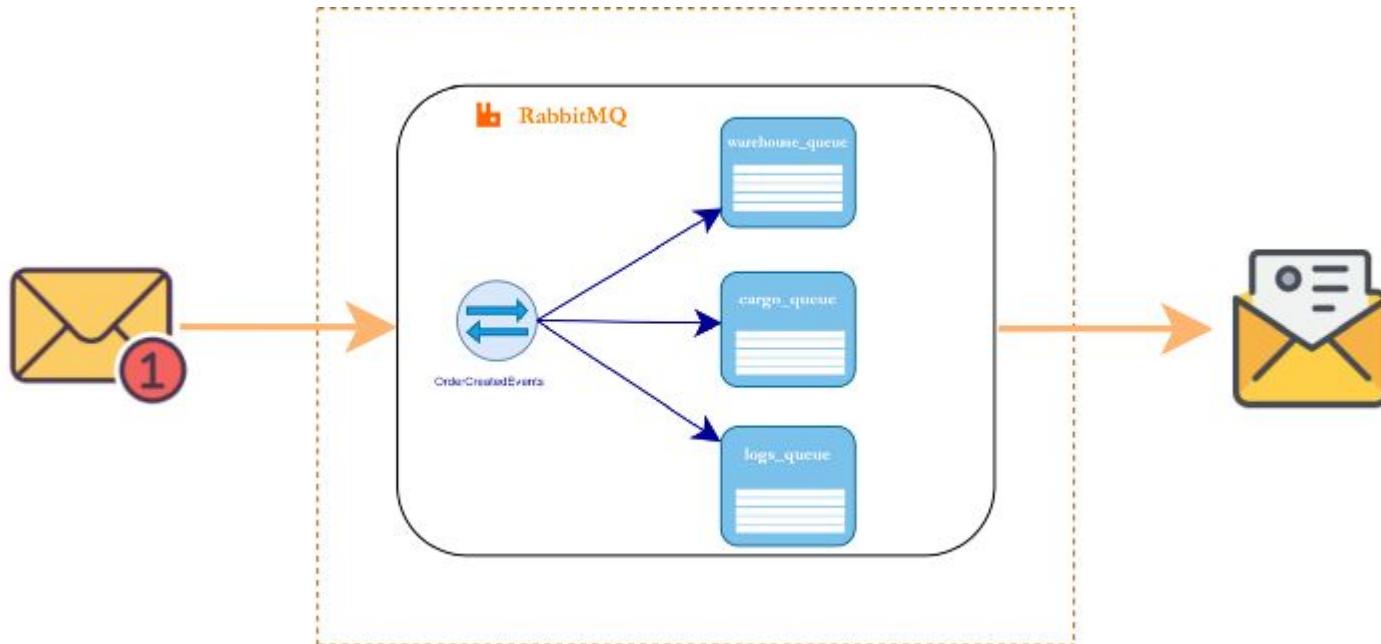


RabbitMQ exchange FANOUT



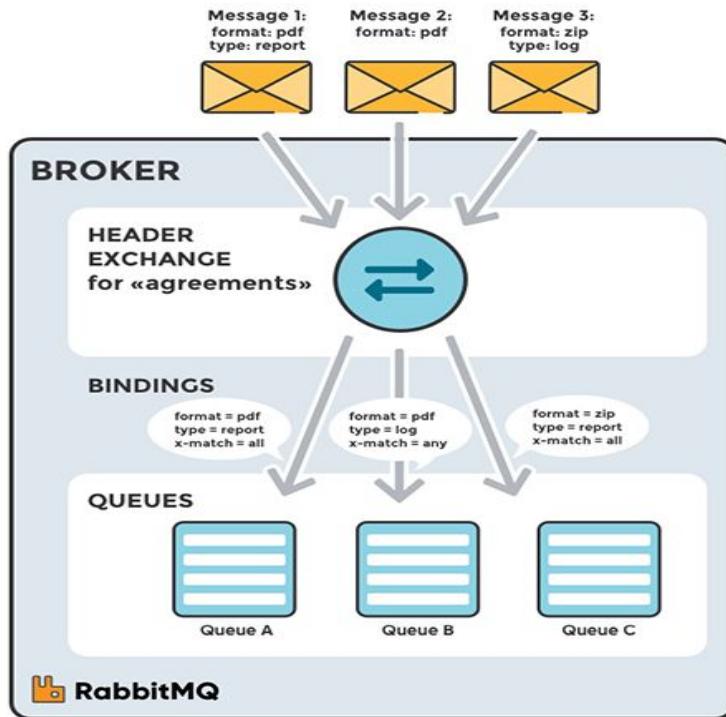


RabbitMQ exchange FANOUT



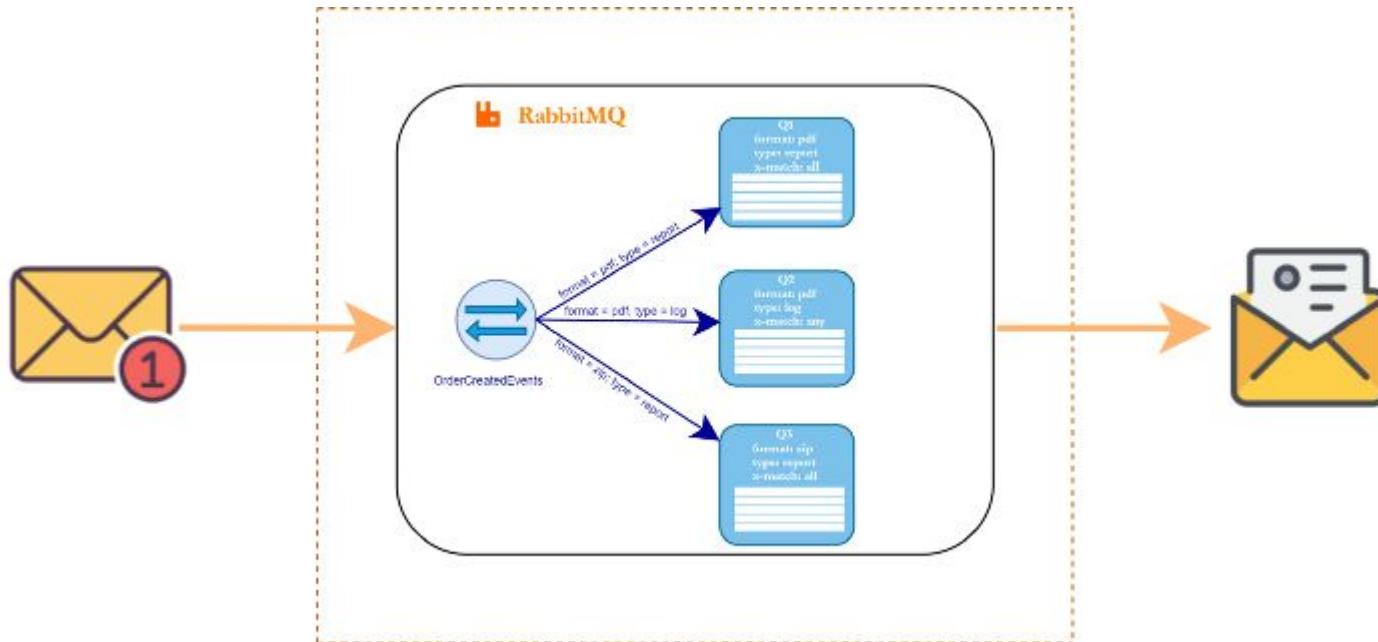


RabbitMQ exchange HEADER



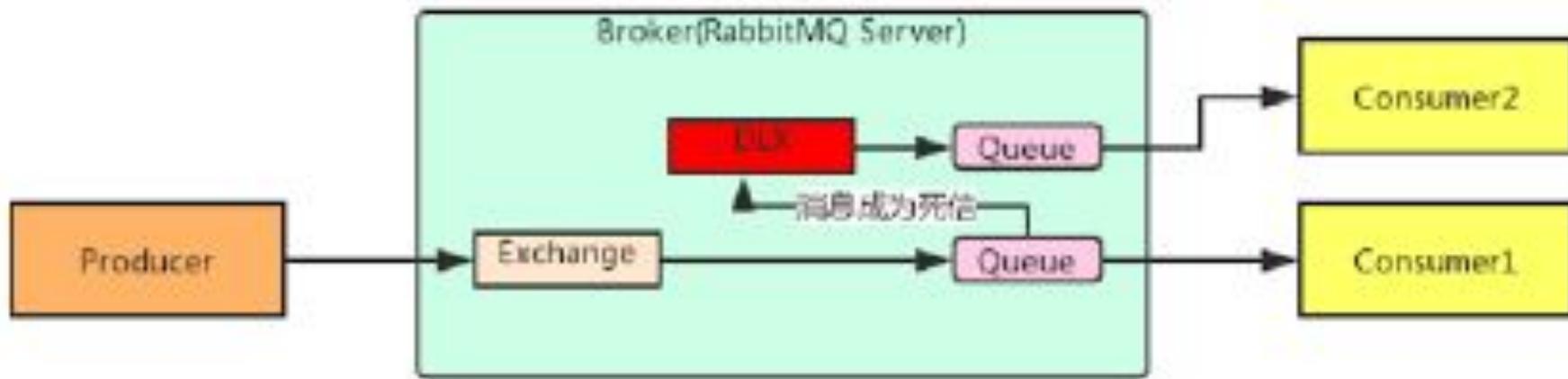


RabbitMQ exchange HEADER



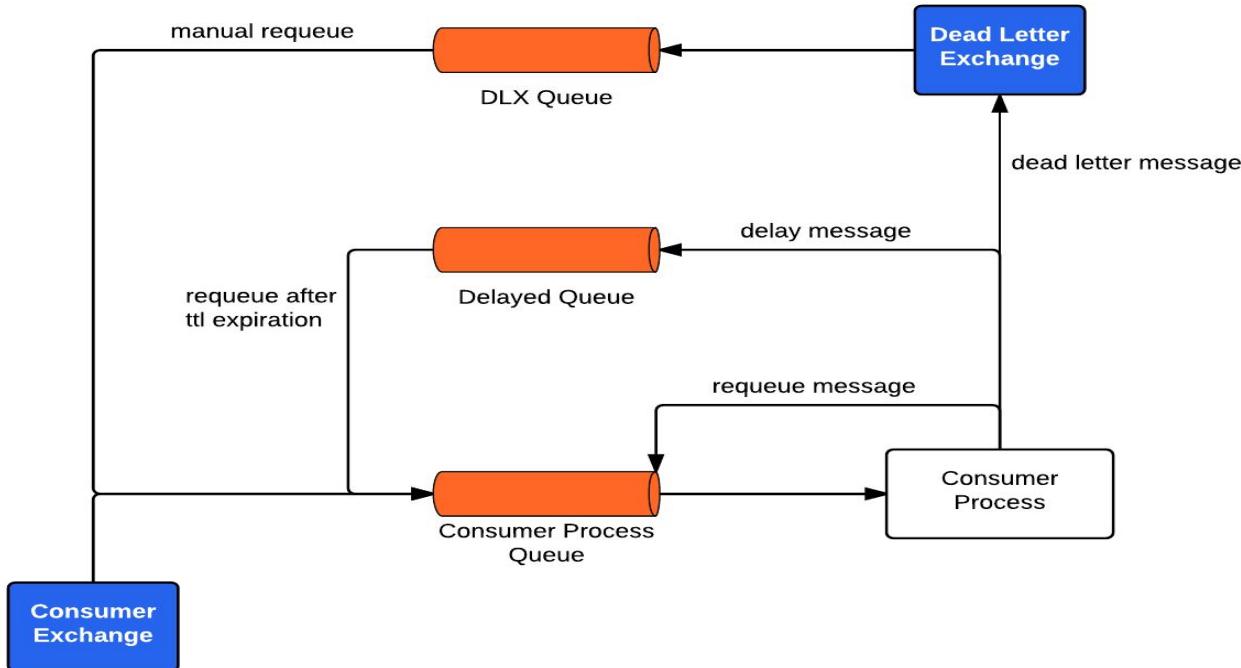


RabbitMQ Dead letter



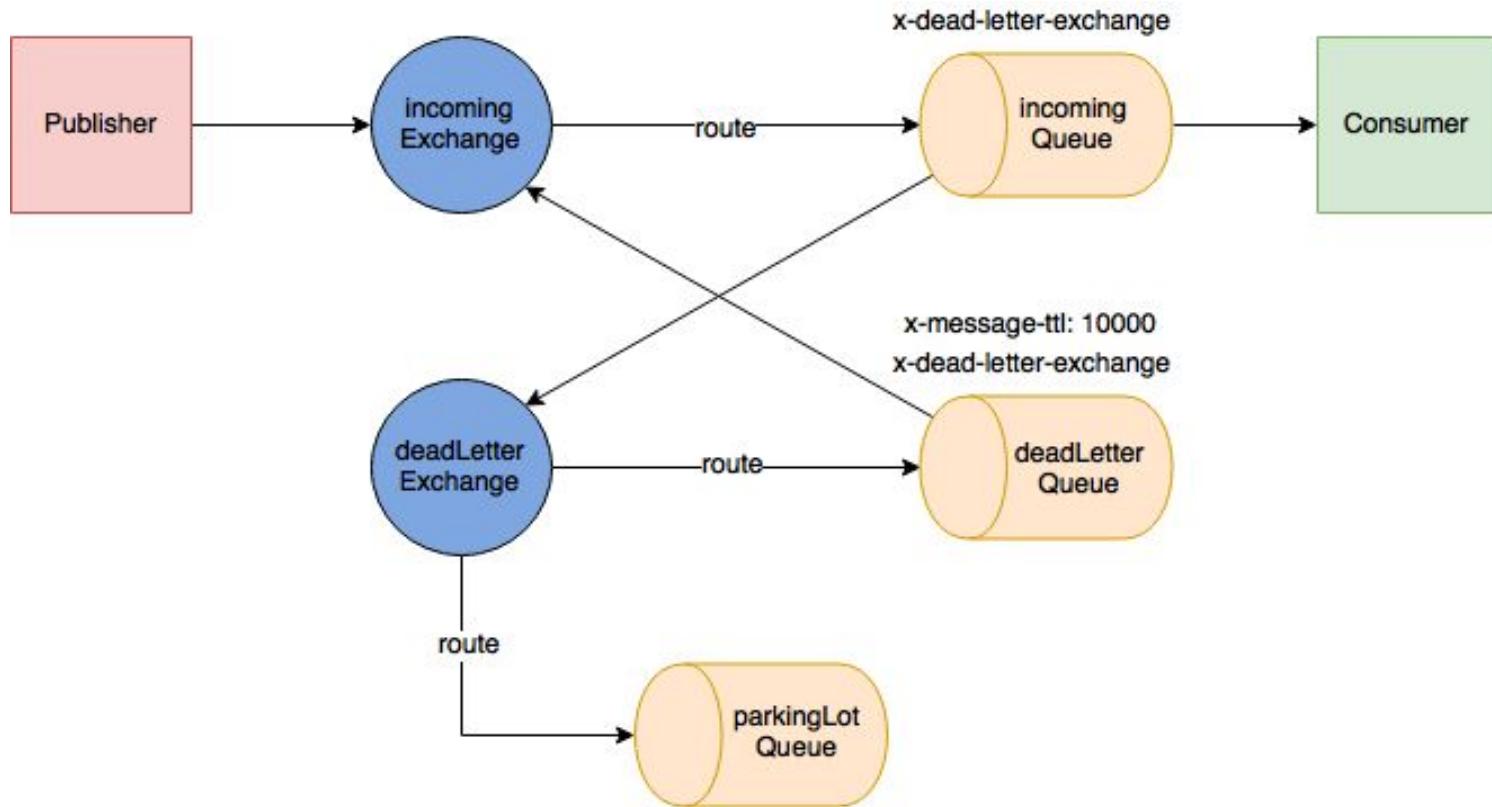


RabbitMQ Dead letter





RabbitMQ TTL (Time to live)





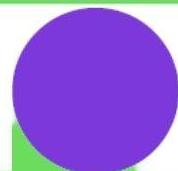
RabbitMQ

 RabbitMQ


docker

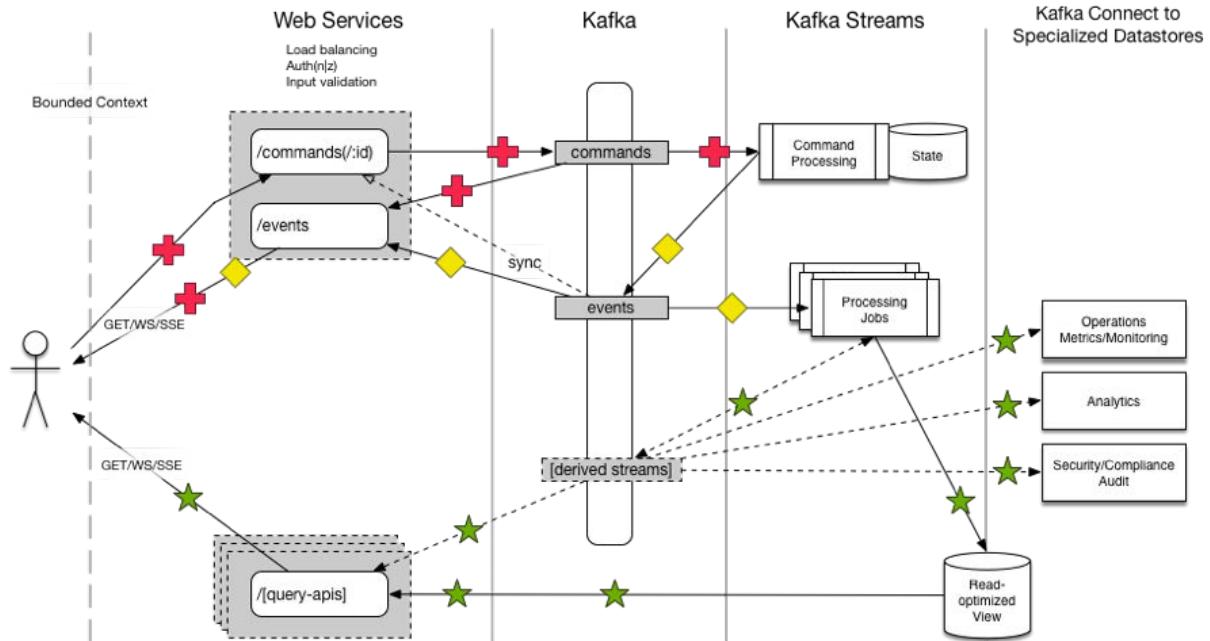
Arquitetura Reativa

Eventos



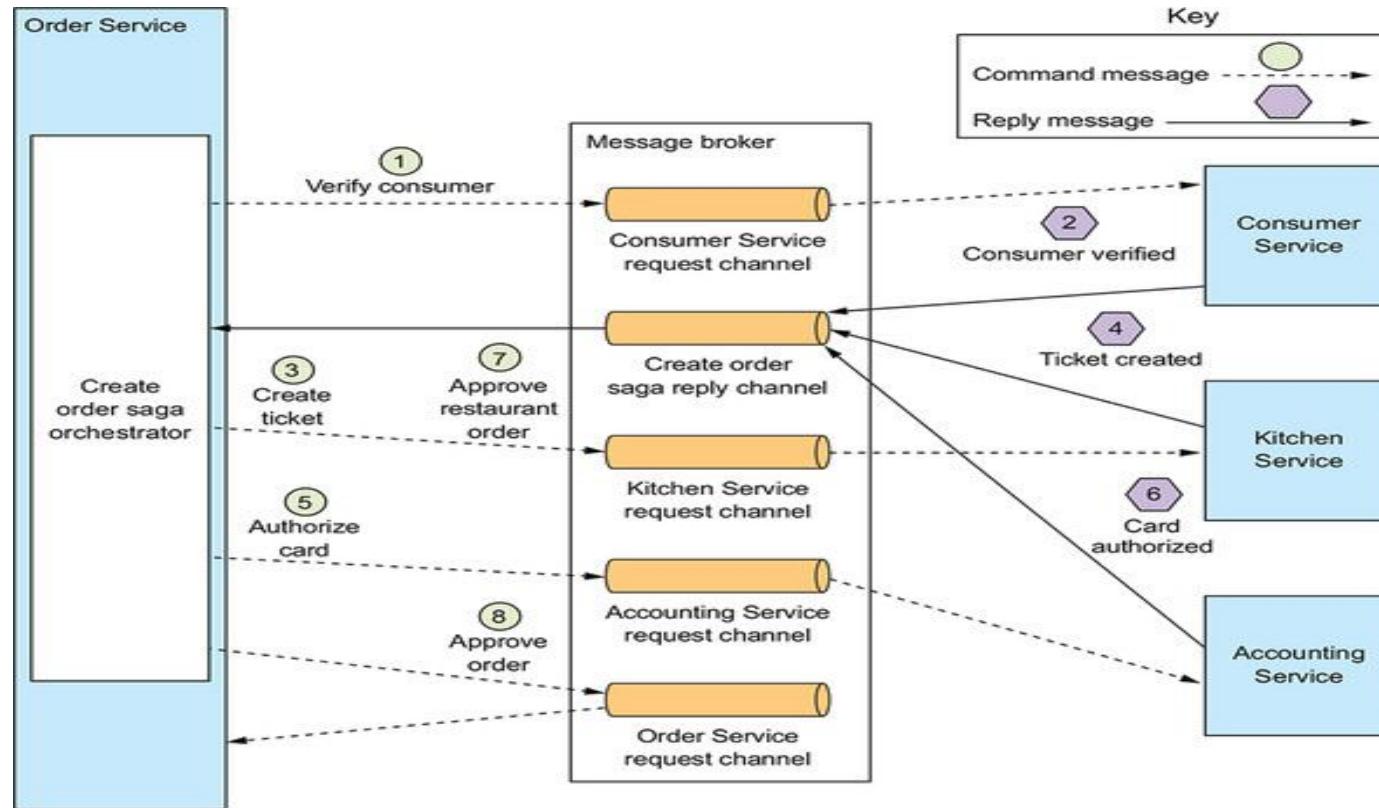


arquitetura reativa



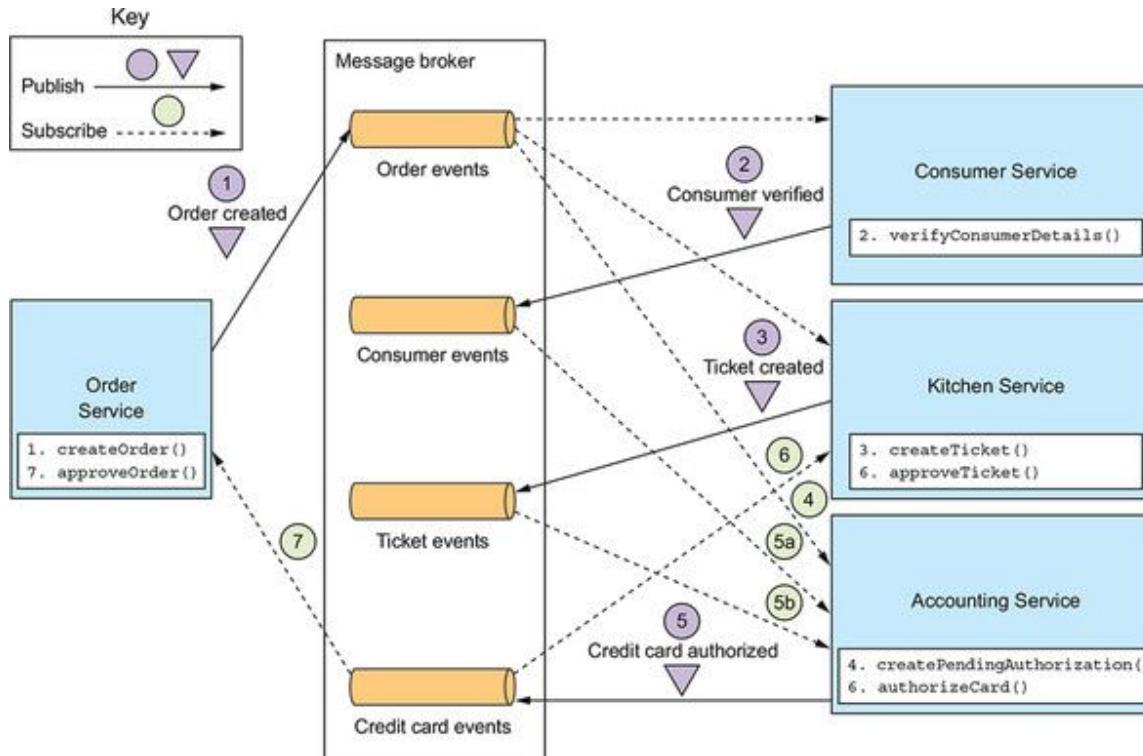


Orquestração



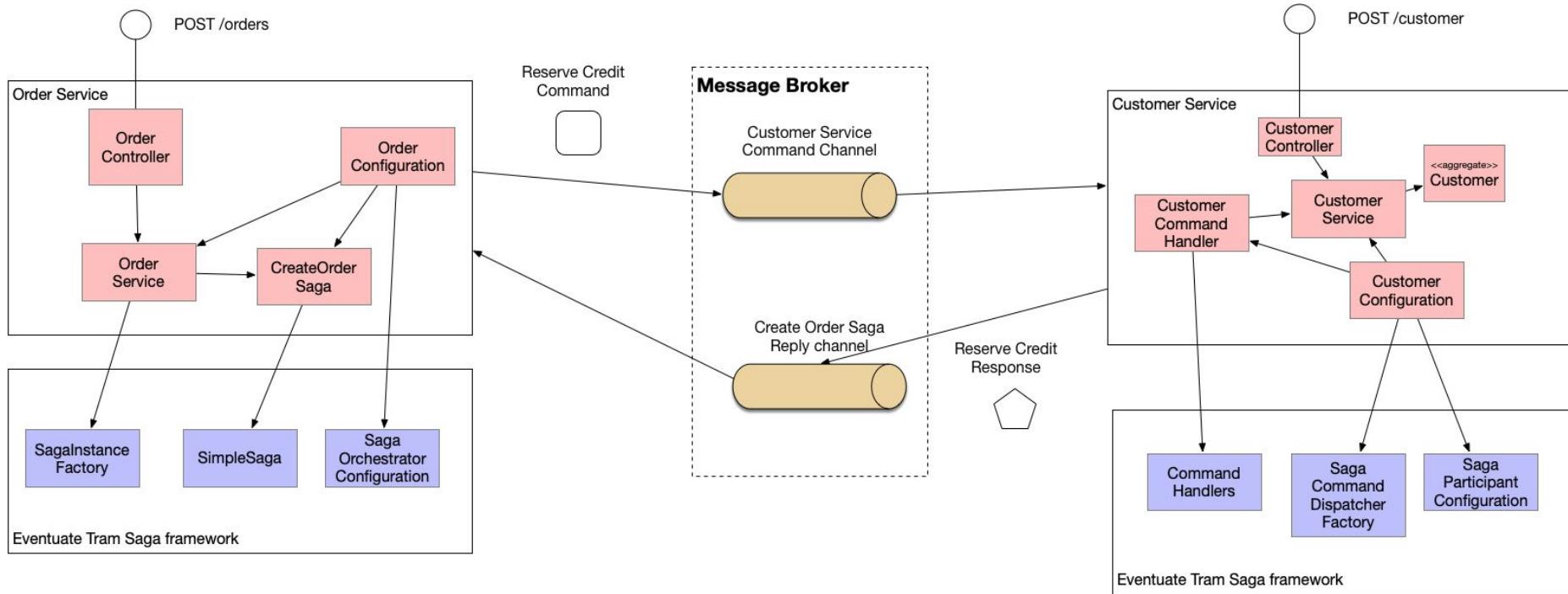


Coreografia





arquitetura reativa



JDBC

Acesso ao banco de **dados**



JDBC framework para acesso a dados

Os bancos relacionais são uma realidade já há um bom tempo. Desta forma, é essencial que qualquer linguagem de programação de peso faça acesso a esses bancos. Foi pensando nisso que a Sun Microsystems introduziu a API JDBC (Java Database Connectivity) no JDK. Este artigo tem o objetivo de mostrar o uso desta tecnologia: como consultar bancos de dados, executar alterações e stored procedures, e muito mais.



Como o JDBC funciona?

JDBC é semelhante ao ODBC, e no princípio usava justamente ODBC para conectar-se com o banco de dados. A partir de um código nativo as aplicações Java podiam utilizar qualquer banco de dados que tivesse um driver ODBC disponível. Isso contribuiu bastante para a **popularização do JDBC** uma vez que existe um driver ODBC para praticamente qualquer banco de dados de mercado.

Assim como ODBC, JDBC também funciona através de drivers que são responsáveis pela conexão com o banco e execução das instruções SQL.



Esses drivers foram divididos em quatro tipos

JDBC tipo 1: foi o primeiro tipo a ser criado, não faz uma conexão real com o banco de dados, mas sim uma conexão com ODBC. Não é muito utilizado hoje em dia por ser escrito em linguagem nativa, o que sacrifica a portabilidade e exige configuração extra no cliente. Ele é composto pelas classes do pacote sun.jdbc.odbc e uma biblioteca de código nativo (não é necessário acessá-la diretamente). Não utilize esse tipo de driver caso tenha outra opção.



Esses drivers foram divididos em quatro tipos

JDBC tipo 2: esse tipo de driver eliminou a dependência de ODBC, mas ainda é escrito em linguagem nativa, esse código nativo permite fazer chamadas a uma API cliente do SGBD. Também é necessária a instalação de bibliotecas de código nativo na máquina onde o sistema será executado, assim como o tipo 1. Esse tipo de driver também não é portável.



Esses drivers foram divididos em quatro tipos

JDBC tipo 3: totalmente escrito em [Java](#), eliminou a necessidade de bibliotecas de código nativo favorecendo a portabilidade. Esse tipo de driver permite a conversão de chamadas JDBC em chamadas a um protocolo de rede genérico que então pode ser convertido a chamadas à API específica do [SGBD](#).

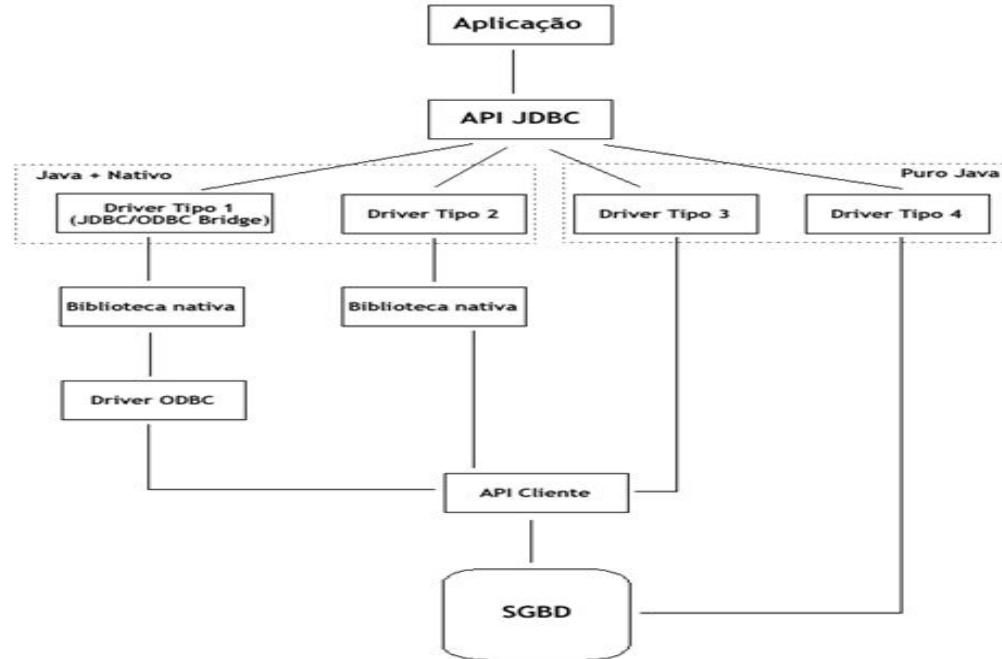


Esses drivers foram divididos em quatro tipos

JDBC tipo 4: também totalmente escrito em Java, utiliza o protocolo de rede proprietário do SGBD, convertendo as chamadas JDBC para chamadas diretas ao SGBD dispensando uma API cliente intermediaria.



Esses drivers foram divididos em quatro tipos





Esses drivers foram divididos em quatro tipos

Esses drivers são implementações das interfaces do pacote `java.sql`. Geralmente são disponibilizados na forma de um arquivo JAR (Java ARchive) pelo fabricante do banco de dados ou terceiros. Você pode **encontrar drivers JDBC** em www.oracle.com, www.dev.mysql.com, www.microsoft.com, www.ibm.com, etc. Ou consultar a base de dados de drivers certificados da Sun em <http://developers.sun.com/product/jdbc/drivers>.

Após fazer o download do driver, basta adicioná-lo ao CLASSPATH (ler Nota 1). A partir daí está tudo pronto para você acessar o banco de dados via Java, como veremos nos próximos slides.

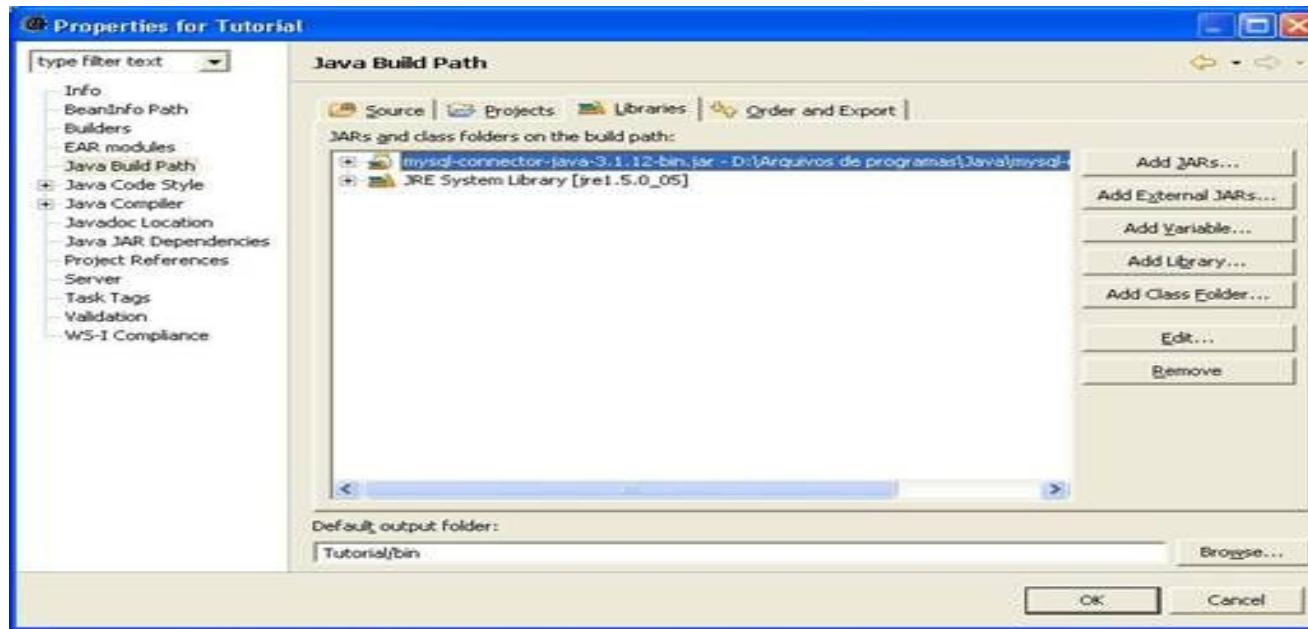


Nota: A maioria dos IDE's ignoram a variável de ambiente CLASSPATH e utilizam uma forma própria de gerenciar as classes usadas pelo projeto (nesse caso é necessário adicionar o driver ao projeto).



Configurando classpath na IDE

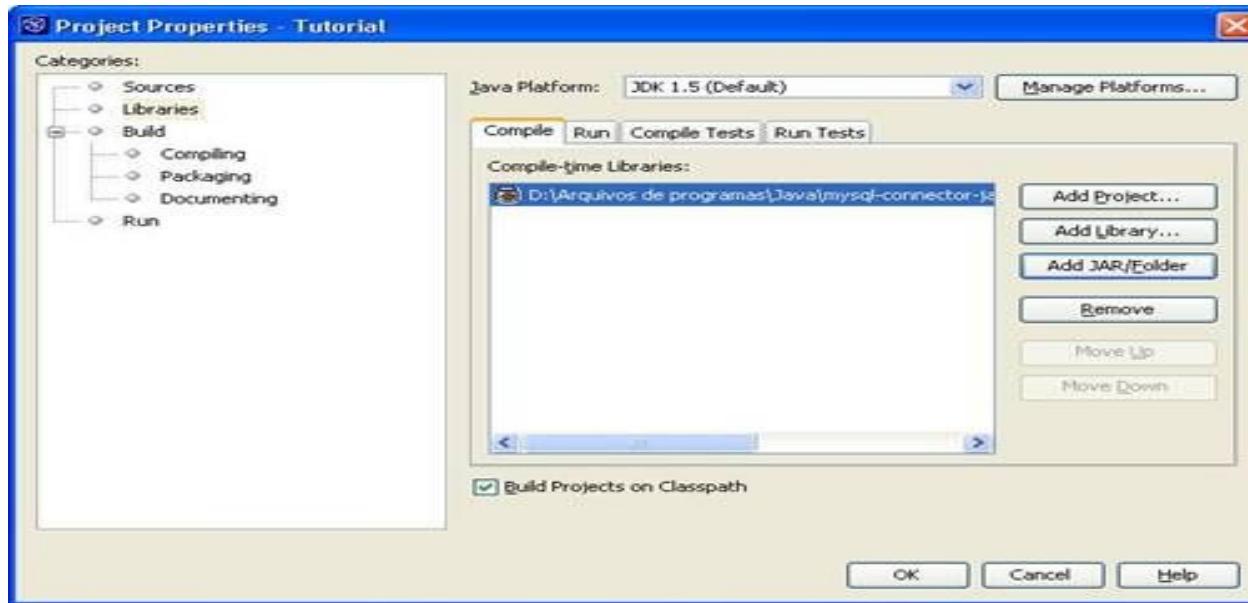
No eclipse : clique com o botão direito no projeto, vá em “Properties”, selecione “Java Build Path” na árvore esquerda, clique na aba “Libraries”, clique no botão “Add External JARs”, selecione o arquivo JAR do driver e clique em “OK”.





Configurando classpath na IDE

Para o NetBeans (ver Figura 3): clique com o botão direito no projeto, vá em “Properties”, selecione “Libraries” na árvore esquerda, clique na aba “Compile”, clique no botão “Add JAR/Folder”, selecione o arquivo JAR do driver e clique em “OK”.





Carregando o Driver

Como vimos, JDBC é baseada em drivers. Para funcionar, esses drivers precisam estar carregados na memória. Quem gerencia esse carregamento é a classe `java.sql.DriverManager`: ao ser instanciado, o driver se registra nela e a partir daí já podem ser criadas conexões utilizando-o. Os métodos `registerDriver(Driver driver)` e `deregisterDriver(Driver driver)` são utilizados para registrar ou remover o registro de um driver, mas a menos que você esteja desenvolvendo seu próprio driver, não é necessário preocupar-se com esses métodos.

Para carregar um driver utilize o método `forName(String name)` da classe `java.lang.Class`. Esse método solicita ao `ClassLoader` o carregamento da classe especificada por “name”, no nosso caso ficaria assim:

```
1 | try{
2 |
3 |     Class.forName("com.mysql.jdbc.Driver");
4 |
5 | }catch(ClassNotFoundException cnfe){
6 |
7 |     cnfe.printStackTrace();
8 |
9 | }
```



Obtendo conexão

Uma conexão é representada pela interface `java.sql.Connection`. É necessário um objeto de uma classe que implemente essa interface (essa classe é fornecida pelo driver, mas você não precisa se preocupar com ela).

A classe `DriverManager` possui alguns métodos `getConnection()`, que são responsáveis por procurar dentre os drivers carregados um que seja compatível com a URL fornecida e solicitar a ele que abra uma conexão.

A URL à qual me referi acima é basicamente o caminho do banco de dados, e tem o seguinte formato:

O protocolo é `jdbc`, o “subprotocolo” e os parâmetros são específicos do driver e geralmente são especificados na documentação. No caso do MySQL, a URL ficaria como a seguir:

Onde:

```
1 | jdbc:mysql://localhost:3306/meu_banco
```

- `jdbc` é o protocolo;
- `mysql`, o sub-protocolo;
- `//localhost` é o endereço do servidor (IP ou nome);
- `3306` é a porta, que é obrigatória caso não seja a padrão e opcional caso seja, e;
- `meu_banco` é nome do banco de dados.



Obtendo conexão

Desta forma, podemos obter uma conexão como apresentado na Listagem 1. O código está dentro de um bloco try porque o método `getConnection()` pode lançar `SQLException`. Essa é a exceção lançada pela maioria dos métodos da API JDBC e pode indicar vários problemas na comunicação com o banco de dados, como erros na expressão SQL, falhas de autenticação e outros.

```
1  try{
2
3      String url = "jdbc:mysql://localhost/tutorial1";
4
5      String usuario = "root";
6
7      String senha = "password";
8
9      Connection conexao = DriverManager.getConnection(url, usuario, senha);
10
11 }catch(SQLException sqle){
12
13     sqle.printStackTrace();
14
15 }
```



Obtendo conexão

O método `getConnection()` procurará por um driver compatível com o formato da URL passada. Caso não encontre lançará uma exceção:

```
1 | java.sql.SQLException: No suitable driver.
```

Caso essa exceção seja lançada no seu código, verifique se o driver foi corretamente carregado. A partir de agora temos aberta uma conexão com o banco de dados e você pode manipulá-la através do objeto `Connection` criado. No slide posterior estão alguns métodos da interface `Connection` que são mais utilizados



Métodos úteis

- **close()**: fecha a conexão.
- **commit()**: realiza um commit em todas as alterações desde o último commit/rollback. Caso a conexão esteja em modo auto-commit não é necessário chamá-lo explicitamente, pois será executado a cada alteração.
- **createStatement()**: um dos métodos mais importantes da conexão, ele cria um objeto Statement que será usado para enviar [expressões SQL](#) para o banco. O retorno é um objeto da interface java.sql.Statement.
- **getMetaData()**: busca os metadados do banco de dados. Metadados seriam basicamente a estrutura do banco, nomes de tabelas, campos, tipos, etc. Retorna um objeto da interface java.sql.DatabaseMetaData.
- **isClosed()**: verifica se a conexão está fechada (retorna true se estiver fechada e false se estiver aberta).
- **isReadOnly()**: verifica se a conexão é somente leitura (retorna true se for somente leitura e false se permitir alterações).
- **prepareCall(String sql)**: cria um objeto para execução de stored procedures, o objeto retornado implementa java.sql.CallableStatement.
- **prepareStatement(String sql)**: Cria um objeto semelhante ao criado por createStatement(), porém permite trabalhar com queries parametrizadas.
- **rollback()**: desfaz as alterações feitas desde o último commit/rollback, é o inverso de commit. Caso a conexão esteja em modo auto-commit não é possível usá-lo, pois a conexão não deixa transações não confirmadas que possam ser desfeitas.
- **setAutoCommit(boolean autoCommit)**: altera o modo auto-commit da conexão (true para ativar e false para desativar). Caso o auto-commit seja desativado, é necessária a chamada explícita ao método commit(), caso contrário as alterações não terão efeito.

Esses são os métodos mais usados da interface Connection, para conhecer outras possibilidades consulte a documentação da API.



Executando consultas

A execução de consultas e atualizações no banco de dados gira em torno da interface `java.sql.Statement` (e sub-interfaces). Para criar um objeto desse tipo, utilize os métodos da conexão que está aberta. Por exemplo:

```
1 | Statement stmt = conexao.createStatement(); //conexao é o nome da variável que criamos acima
```

Com uma instância de `Statement` já podemos executar uma query no banco, como abaixo:

```
1 | ResultSet resultado = stmt.executeQuery("select * from clientes");
```

O método `executeQuery()` é usado para executar consultas apenas, e não deve ser usado para comandos como `update`, `delete`, `create`, etc. Para isso temos o método `executeUpdate()` que veremos adiante. Já o método `execute()` é utilizado em situações em que a query pode retornar mais de um resultado (somente em situações muito particulares ele é utilizado, como em algumas execuções de stored procedures).



Obtendo cursor com ResultSet

O resultado da consulta está no objeto “resultado” da interface `java.sql.ResultSet`. Esse objeto possui uma série de métodos `getXXX()` usados para recuperar os dados que estão no objeto `ResultSet` de acordo com um tipo. Por exemplo, para recuperar uma coluna do tipo `int` use o método `getInt()`, como `String getString()`, como `double getDouble()`, e assim por diante.

Um `ResultSet` controla a posição dos registros retornados utilizando um ponteiro. Esse ponteiro aponta para uma determinada linha, chamada de `current`, de onde serão retirados os dados ao chamar um dos métodos `get`. Ao ser criado, esse ponteiro aponta para uma linha antes da primeira válida, e é necessário movê-lo para uma linha válida antes de acessar os dados, caso contrário será lançada a exceção:

```
1 | java.sql.SQLException: Before start of ResultSet
```



Métodos ResultSet

Para mover esse ponteiro, use um dos métodos de navegação do ResultSet:

- **absolute(int row)**: move para uma linha específica;
- **afterLast()**: move para a linha após a última, ou seja uma linha inválida;
- **beforeFirst()**: move para a linha antes da primeira, exatamente como quando o ResultSet é criado, também uma linha inválida;
- **first()**: move para a primeira linha;
- **last()**: move para a última linha;
- **next()**: move para a próxima linha;
- **previous()**: move para a linha anterior;
- **relative(int rows)**: move algumas posições, especificadas pelo parâmetro, relativamente à atual.



Métodos ResultSet

No nosso exemplo, podemos ter algo da seguinte forma:

```
1 | resultado.first();
2 |
3 | String nome = resultado.getString("nome"); //recupera o valor da coluna 'nome' como String
4 |
5 | String cnpj = resultado.getString(2); //recupera o valor da segunda coluna como String
6 |
7 | Date cadastro = resultado.getDate("cadastro");
```



Métodos ResultSet

O código da Listagem 2 apresentará todos os registros de uma tabela ‘clientes’ (código, nome, cnpj).

```
1 try{  
2     Class.forName("com.mysql.jdbc.Driver");  
3     //carrega o driver  
4  
5     Connection conexao = DriverManager.getConnection("jdbc:mysql://localhost/meuBanco", "root"  
6     //obtém uma conexão  
7  
8     Statement stmt = conexao.createStatement();  
9     //cria um Statement  
10  
11    ResultSet resultado = stmt.executeQuery("select * from clientes");  
12    //executa uma consulta  
13  
14}
```



Métodos ResultSet

```
14  
15  
16  
17     while(resultado.next()){ //o método next() retorna true caso haja mais linhas  
18         System.out.print(resultado.getInt("codigo")+"\\t");  
19  
20         System.out.print(resultado.getString("nome")+"\\t");  
21  
22         System.out.println(resultado.getString("cnpj"));  
23  
24     }  
25  
26 }
```



Métodos ResultSet

```
27  
28  
29     resultado.close(); //fecha o ResultSet  
30  
31     stmt.close(); //fecha o Statement  
32  
33     conexao.close(); //encerra a conexão  
34  
35  
36  
37 }catch(SQLException sqle){  
38  
39     sqle.printStackTrace();  
40  
41 }catch(Exception e){  
42  
43     e.printStackTrace();  
44  
45 }
```



Métodos ResultSet

O resultado seria:

1	1	Jair	00.000.000/0001-00
2			
3	2	Acme	11.111.111/1111-11



Métodos ResultSet

Por padrão, um Statement só permite um ResultSet aberto por vez. Para manter vários ResultSets abertos simultaneamente, eles devem ser criados por Statements diferentes. Por exemplo:

```
1 | Statement stmt = conexao.createStatement();
2 |
3 | ResultSet resultado1 = stmt.executeQuery("select * from clientes");
4 |
5 | ResultSet resultado2 = stmt.executeQuery("select * from produtos");
6 |
7 | resultado1.next();
```



Métodos ResultSet

Entretanto, a última linha desse código lançará a seguinte exceção:

```
1 | java.sql.SQLException: Operation not allowed after ResultSet closed
```

O primeiro ResultSet foi fechado ao executar o método `executeQuery()` pela segunda vez, pois implicitamente todos os ResultSets criados pelo mesmo Statement são fechados. O seguinte código resolve o problema:

```
1 | Statement stmt = conexao.createStatement();
2 |
3 | ResultSet resultado1 = stmt.executeQuery("select * from clientes");
4 |
5 | Statement stmt2 = conexao.createStatement();
6 |
7 | ResultSet resultado2 = stmt2.executeQuery("select * from produtos");
8 |
9 | resultado1.next();
```



Métodos ResultSet

Também pode ser usado o método getMoreResults(int current) para manter os ResultSets abertos.

```
1 Statement stmt = conexao.createStatement();
2
3 ResultSet resultado1 = stmt.executeQuery("select * from clientes");
4
5 stmt.getMoreResults(Statement.KEEP_CURRENT_RESULT);
6
7 ResultSet resultado2 = stmt.executeQuery("select * from produtos");
8
9 resultado2.next();
```



Executando alterações

Como foi dito anteriormente, o método `executeQuery()` só é válido para consultas. Para executar alterações utilize um dos métodos `executeUpdate()` disponíveis na interface Statement.

A forma mais simples do método `executeUpdate()` recebe uma String que é a expressão SQL a ser executada. Essa expressão não pode retornar um resultado e deve ser um INSERT, UPDATE ou DELETE. O retorno do método é um valor int que indica o número de linhas afetadas.



Executando alterações

O código da Listagem 3 insere um registro na tabela ‘clientes’ mencionada anteriormente:

```
1 try{
2
3     Class.forName("com.mysql.jdbc.Driver"); //carrega o driver
4
5     Connection conexao = DriverManager.getConnection("jdbc:mysql://localhost/meuBanco",
6             "root", "senha"); //obtém uma conexão
7
8     Statement stmt = conexao.createStatement(); //cria um Statement
9
10    int linhas = stmt.executeUpdate("insert into clientes (nome, cnpj) values
11        ('Acme Corporation', '00.000.000/0001-00')"); //insere o registro
12
13
14    //caso a conexão estivesse com auto-commit desativado nesse ponto seria
15    //necessário chamar conexao.commit();
16
17
18
19
20    System.out.println(linhas+" linhas afetadas!");
```



Executando alterações

A maioria dos [bancos de dados](#) possui algum tipo de geração automática para certos campos, (auto_increment no MySQL). Após inserir um registro pode ser necessário obter o valor gerado nesses campos, para isso utilize o método getGeneratedKeys() do Statement

```
1 | try{
2 |
3 |     Class.forName("com.mysql.jdbc.Driver");
4 |     //carrega o driver
5 |
6 |     Connection conexao = DriverManager.getConnection("jdbc:mysql://localhost/meuBanco", "root"
7 |     //obtém uma conexão
8 |
9 |     Statement stmt = conexao.createStatement();
10 |    //cria um Statement
11 |
12 |    stmt.executeUpdate("insert into clientes (nome, cnpj) values ('Foo Bar', '00.000.000/0001-0
13 |    //insere o registro
14 |
15 |
16 |
17 |    ResultSet res = stmt.getGeneratedKeys(); //obtém as chaves geradas
```



Executando alterações

```
16  
17     ResultSet res = stmt.getGeneratedKeys(); //obtém as chaves geradas  
18  
19  
20  
21     if(res.first()){ //caso o método first() retorne false, não há registros no ResultSet  
22         System.out.println("Código gerado: "+res.getInt(1));  
23     }else{  
24         System.out.println("Nenhum código gerado!");  
25     }  
26  
27  
28  
29  
30  
31  
32  
33     stmt.close(); //fecha o Statement  
34  
35     conexao.close(); //encerra a conexão  
36
```



ResultSet atualizável

É possível utilizar o próprio ResultSet para atualizar os dados na tabela. Para isso é necessário criar o ResultSet como atualizável passando a constante ResultSet.CONCUR_UPDATABLE na criação do Statement que dará origem ao ResultSet:

```
1 | Statement stmt = conexao.createStatement	ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPD
2 |
3 | ResultSet resultado = stmt.executeQuery("select * from clientes");
```



ResultSet atualizável

A partir do objeto ResultSet criado é possível utilizar um dos métodos updateXXX() para alterar valores da linha atual. Por exemplo:

```
1 | resultado.first();
2 |
3 | resultado.updateString("nome", "Empresa Ficticia");
4 |
5 | resultado.updateString("cnpj", "12.123.123/0001-12");
6 |
7 | resultado.updateRow();
```



ResultSet atualizável

Também é possível inserir novos registros a partir do ResultSet. Utilizando o método `moveToInsertRow()` é possível mover-se para uma linha “em branco” do ResultSet, colocar dados nela e utilizar o método `insertRow()` para gravar as alterações no banco de dados.

```
1 | resultado.moveToInsertRow();
2 |
3 | resultado.updateString("nome", "Novo Cliente");
4 |
5 | resultado.updateString("cnpj", "55.555.555/0001-55");
6 |
7 | resultado.insertRow();
```

Caso você desista de inserir o registro e queira voltar à linha normal, utilize o método `moveToCurrentRow()`.



ResultSet atualizável

Para excluir uma linha na tabela, pode ser usado o método `deleteRow()`.

```
1 | resultado.first();
2 |
3 | resultado.deleteRow();
```



Atualizações em batch

É possível utilizar o Statement para enviar atualizações em batch (grupo) para o banco de dados. Para isso, basta adicionar as expressões SQL utilizando o método addBatch() e executá-las com o método executeBatch().

```
1 | Statement stmt = conexao.createStatement();
2 |
3 | stmt.addBatch("insert into clientes values(null, 'cliente 1', '00.000.000/0001-00');");
4 |
5 | stmt.addBatch("insert into clientes values(null, 'cliente2', '11.111.111/0001-11');");
6 |
7 | stmt.addBatch("update clientes set nome='Novo nome' where codigo=1");
8 |
9 | int[] linhas = stmt.executeBatch();
```



Atualizações em batch

O retorno do método executeBatch() é um array de int com a quantidade de linhas afetadas em cada expressão executada.

executeBatch() geralmente é executado com o auto-commit desativo, pois caso ocorra um erro em uma das expressões é possível desfazer as alterações . Nesse caso, serão inseridos os dois registros ou nenhum.

```
1  try{
2
3     conexao.setAutoCommit(false);
4
5      Statement stmt = conexao.createStatement();
6
7      stmt.addBatch("insert into clientes values(null, 'cliente 1', '00.000.000/0001-00')");
8
9      stmt.addBatch("insert into clientes values(null, 'cliente2', '11.111.111/0001-11')");
10
11     stmt.executeBatch();
12
13     conexao.commit();
14
15 }catch(BatchUpdateException bue){
16
17     conexao.rollback();
18 }
19 }
```



PreparedStatement

Como as expressões SQL são tratadas como strings, para inserir um valor nela é necessário concatenar valores. Isso pode não ser muito conveniente, então podemos usar uma sub-interface de Statement para trabalhar com expressões parametrizadas, o PreparedStatement.

Ao ser criada, a expressão SQL é enviada ao banco e compilada, assim pode ser usada diversas vezes somente mudando os parâmetros, o que pode aumentar significativamente o desempenho.

Um objeto PreparedStatement é obtido através do método `prepareStatement(String sql)` da conexão, o parâmetro que será passado é a expressão SQL e no lugar de cada parâmetro um ?:

```
1 | PreparedStatement stmt = conexao.prepareStatement("insert into amigos  
2 | (nome, telefone) values (?, ?);");
```



PreparedStatement

A partir desse objeto é possível inserir vários registros na tabela ‘amigos’ simplesmente setando os parâmetros:

```
1  stmt.setString(1, "João");
2
3  stmt.setString(2, "(31) 5555-5555");
4
5  stmt.executeUpdate();
6
7  stmt.setString(1, "Ana");
8
9  stmt.setString(2, "(31) 1111-1111");
10
11 stmt.executeUpdate();
```



PreparedStatement

Veja que não é necessário recriar o objeto para executar outra alteração, os dados dos parâmetros são retidos no objeto até que sejam substituídos por outros, ou pela chamada do método `clearParameters()`.

```
1  stmt.setString(1, "José");
2
3  stmt.setString(2, "(31) 9999-9999");
4
5  stmt.executeUpdate();
6
7  stmt.clearParameters();
8
9  stmt.setString(1, "Maria");
10
11 stmt.setString(2, "(11) 1234-1234");
12
13 stmt.executeUpdate();
```



Stored procedures

A chamada de stored procedures também é possível com JDBC. A interface base para isso é `java.sql.CallableStatement` (sub-interface de `PreparedStatement`).

Assim como um `Statement` ou `PreparedStatement`, um `CallableStatement` é criado a partir de uma conexão ativa através do método `prepareCall(String sql)`. A String passada como parâmetro tem uma sintaxe particular e padronizada, o driver é responsável por convertê-la para a sintaxe específica do SGBD utilizado.

Para exemplificar, vamos emitir a listagem de clientes utilizando uma stored procedure. O procedimento seria criado conforme

```
2 | CallableStatement chamada = conexao.prepareCall("{call sp_listarClientes()}"); //monta a
3 |
4 |
5 | ResultSet resultado = chamada.executeQuery(); //executa
6 |  
7 |  
8 |  
9 |
```

JPA

Acesso ao banco de **dados**



JPA

JPA é um framework leve, baseado em [POJOS \(Plain Old Java Objects\)](#) para persistir objetos Java. A **Java Persistence API**, diferente do que muitos imaginam, não é apenas um framework para [Mapeamento Objeto-Relacional \(ORM - Object-Relational Mapping\)](#), ela também oferece diversas funcionalidades essenciais em qualquer aplicação corporativa.

Atualmente temos que praticamente todas as aplicações de grande porte **utilizam JPA para persistir objetos Java**. JPA provê diversas funcionalidades para os programadores, como será mais detalhadamente visto nas próximas seções. Inicialmente será visto a história por trás da JPA, a qual passou por algumas versões até chegar na sua versão atual.

É uma especificação, e como uma especificação, ela preocupa-se com a persistência, o que significa vagamente qualquer mecanismo pelo qual os objetos Java sobrevivam ao processo do aplicativo que os criou. Nem todos os objetos Java precisam ser persistidos, mas a maioria dos aplicativos persiste os principais objetos de negócios.



História da Especificação

Após diversos anos de reclamações sobre a complexidade na construção de aplicações com Java, a especificação [Java EE 5](#) teve como principal objetivo a facilidade para desenvolver aplicações JEE 5. O EJB 3 foi o grande precursor para essa mudança fazendo os Enterprise JavaBeans mais fáceis e mais produtivos de usar.

No caso dos Session Beans e Message-Driven Beans, a solução para questões de usabilidade foram alcançadas simplesmente removendo alguns dos mais onerosos requisitos de implementação e permitindo que os componentes sejam como Plain Java Objects ou POJOS.

Já os Entity Beans eram um problema muito mais sério. A solução foi começar do zero. Deixou-se os Entity Beans sozinhos e introduziu-se um novo modelo de persistência. A versão atual da JPA nasceu através das necessidades dos profissionais da área e das soluções proprietárias que já existiam para [resolver os problemas com persistência](#). Com a ajuda dos desenvolvedores e de profissionais experientes que criaram outras ferramentas de persistência, chegou a uma versão muito melhor que é a que os desenvolvedores Java conhecem atualmente.

Dessa forma os líderes das soluções de mapeamento objetos-relacionais deram um passo adiante e padronizaram também os seus produtos. Hibernate e TopLink foram os primeiros a firmar com os fornecedores EJB.

O resultado final da especificação EJB finalizou com três documentos separados, sendo que o terceiro era o Java Persistence API. Essa especificação descrevia o modelo de persistência em ambos os ambientes [Java SE](#) e Java EE.



JPA 2.0

No momento em que a primeira versão do JPA foi iniciada, outros modelos de persistência ORM já haviam evoluído. Mesmo assim muitas características foram adicionadas nesta versão e outras foram deixadas para uma próxima versão.

A versão JPA 2.0 incluiu um grande número de características que não estavam na primeira versão, especialmente as mais requisitadas pelos usuários, entre elas a capacidade adicional de mapeamento, expansões para a [Java Persistence Query Language \(JPQL\)](#), a API Criteria para criação de consultas dinâmicas, entre outras características.



JPA 2.0

Entre as principais inclusões na JPA destacam-se:

- **POJOS Persistentes:** Talvez o aspecto mais importante da JPA seja o fato que os objetos são POJOs (Plain Old Java Object ou Velho e Simples Objeto Java), significando que os objetos possuem design simples que não dependem da herança de interfaces ou classes de frameworks externos. Qualquer objeto com um construtor default pode ser feito persistente sem nenhuma alteração numa linha de código. Mapeamento Objeto-Relacional com JPA é inteiramente dirigido a metadados. Isto pode ser feito através de anotações no código ou através de um XML definido externamente.
- **Consultas em Objetos:** As consultas podem ser realizadas através da Java Persistence Query Language (JPQL), uma linguagem de consulta que é derivada do EJB QL e transformada depois para SQL. As consultas usam um esquema abstraído que é baseado no modelo de entidade como oposto às colunas na qual a entidade é armazenada.



JPA 2.0

Entre as principais inclusões na JPA destacam-se:

- **Configurações simples:** Existe um grande número de características de persistência que a especificação oferece, todas são configuráveis através de anotações, XML ou uma combinação das duas. Anotações são simples de usar, convenientes para escrever e fácil de ler. Além disso, JPA oferece diversos valores defaults, portanto para já sair usando JPA é simples, bastando algumas anotações.
- **Integração e Teste:** Atualmente as aplicações normalmente rodam num Servidor de aplicação, sendo um padrão do mercado hoje. Testes em servidores de aplicação são um grande desafio e normalmente impraticáveis. Efetuar teste de unidade e teste caixa branca em servidores de aplicação não é uma tarefa tão trivial. Porém, isto é resolvido com uma API que trabalha fora do servidor de aplicação. Isto permite que a JPA possa ser utilizada sem a existência de um servidor de aplicação. Dessa forma, testes unitários podem ser executados mais facilmente.



JPA 2.0

Entre as principais inclusões na JPA destacam-se:

- **Configurações simples:** Existe um grande número de características de persistência que a especificação oferece, todas são configuráveis através de anotações, XML ou uma combinação das duas. Anotações são simples de usar, convenientes para escrever e fácil de ler. Além disso, JPA oferece diversos valores defaults, portanto para já sair usando JPA é simples, bastando algumas anotações.
- **Integração e Teste:** Atualmente as aplicações normalmente rodam num Servidor de aplicação, sendo um padrão do mercado hoje. Testes em servidores de aplicação são um grande desafio e normalmente impraticáveis. Efetuar teste de unidade e teste caixa branca em servidores de aplicação não é uma tarefa tão trivial. Porém, isto é resolvido com uma API que trabalha fora do servidor de aplicação. Isto permite que a JPA possa ser utilizada sem a existência de um servidor de aplicação. Dessa forma, testes unitários podem ser executados mais facilmente.



JPA 2.0

JPA evoluiu ao longo do tempo e o que influenciou as características presentes na sua versão atual, considerada uma grande evolução na plataforma Java. Além disso, tivemos a oportunidade de verificar quais são as principais inclusões na atual versão da JPA e como eles podem impulsionar o desenvolvimento de aplicações corporativas que exigem uma manutenção mais simples, fácil e rápida além de confiabilidade e testes de unidades, indispensável em qualquer aplicação corporativa hoje. Conforme solicitado pelos usuários, esse primeiro artigo introdutório foi feito pensando nas características básica da JPA, enfatizando como se deu seu desenvolvimento e quais foram as suas principais características disponibilizadas nas últimas versões.



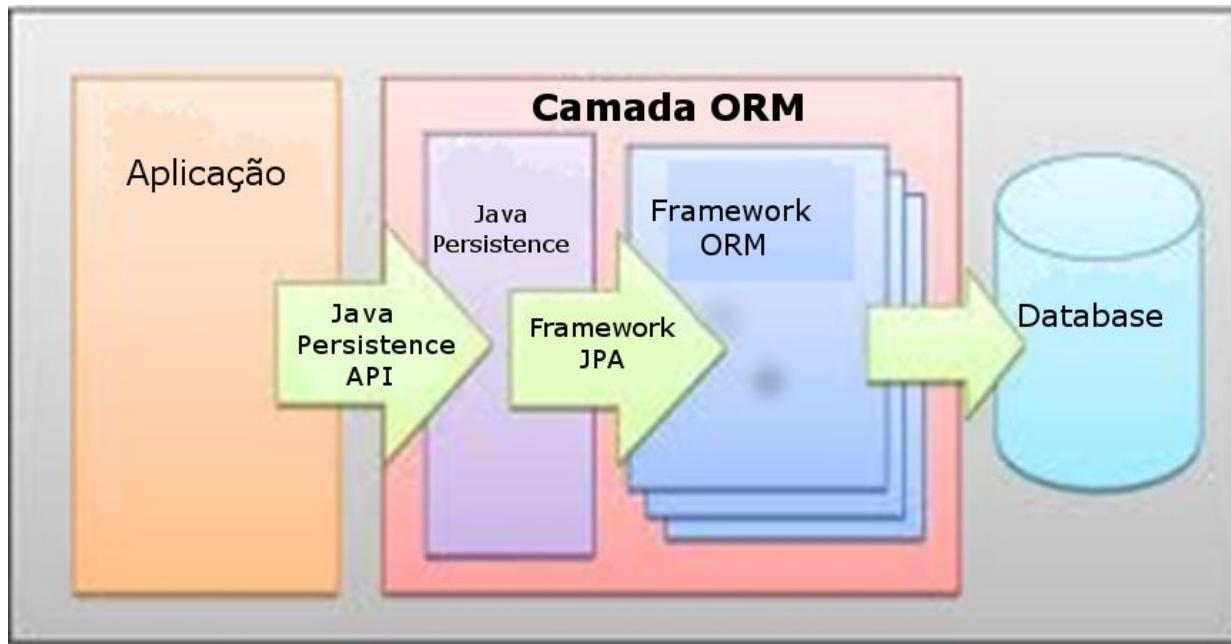
**JPA - Java Persistence API
O que é ?
Como Funciona ?**





JPA 2.0

Abaixo segue um diagrama sobre ORM – Object Relational Mapping (Mapeamento Objeto Relacional)





JPA 2.0

O diagrama mostra o fluxo de funcionamento da API JPA, de onde a ela é inicialmente chamada, e até onde ela vai.

Como o primeiro framework Java para ORM foi o Hibernate, no início eram utilizados arquivos XML, os conhecidos hbm.xml, depois veio o XDoclet, antecessor das Annotations. Era utilizado com ejb's e o Hibernate.

A JPA, é uma especificação Java para acessar, persistir e gerenciar dados entre objetos Java e um banco de dados relacional. O JPA foi definido como parte da especificação EJB 3.0 como um substituto para a especificação EJB 2 CMP Entity Beans. A JPA veio da necessidade de simplificar a complexidade do EJB para persistir dados.

A JPA agora é considerada a abordagem padrão da indústria para ORM – Object to Relational Mapping. Mas independente do framework ORM utilizado, seja o Hibernate, o EclipseLink, o TopLink, o OpenJPA, etc, isso não importa, a aplicação será portável para qualquer banco de dados que possua driver JDBC. E não será preciso reescrever o código-fonte, pois ele será o mesmo para todos os bancos de dados.

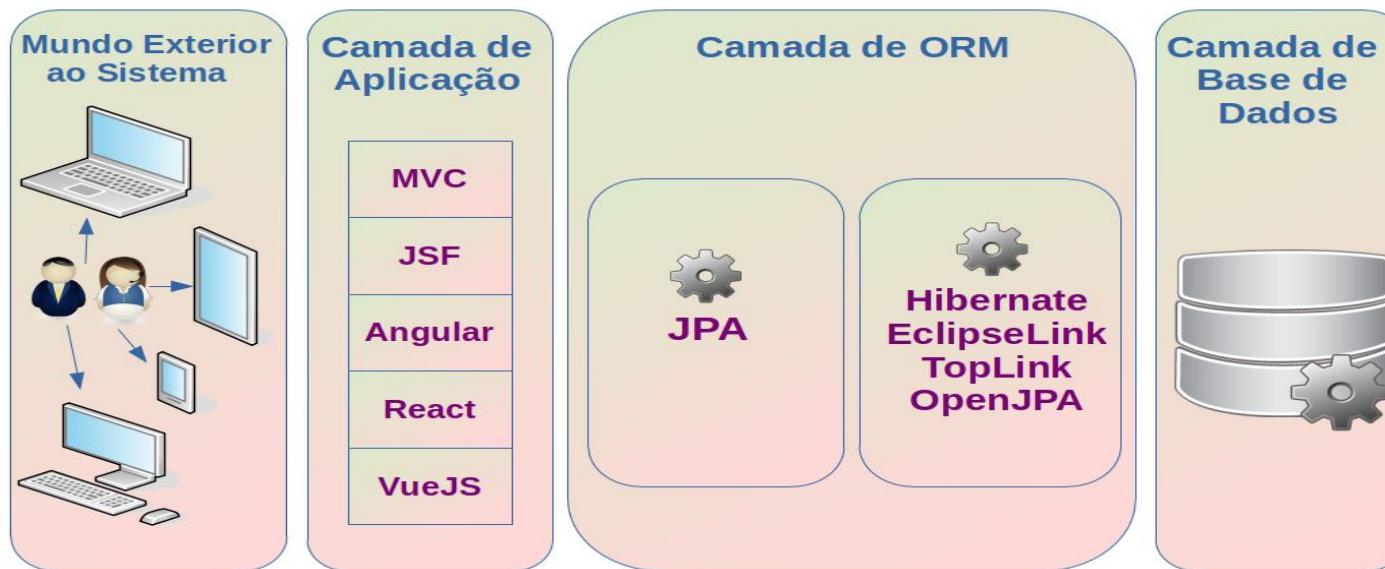


JPA 2.0

Como funciona a JPA – Ele funciona através de qualquer framework ORM (Mapeamento Objeto Relacional) baseado na especificação JPA.

Pode ser o framework Hibernate, EclipseLink, TopLink, OpenJpa, etc.

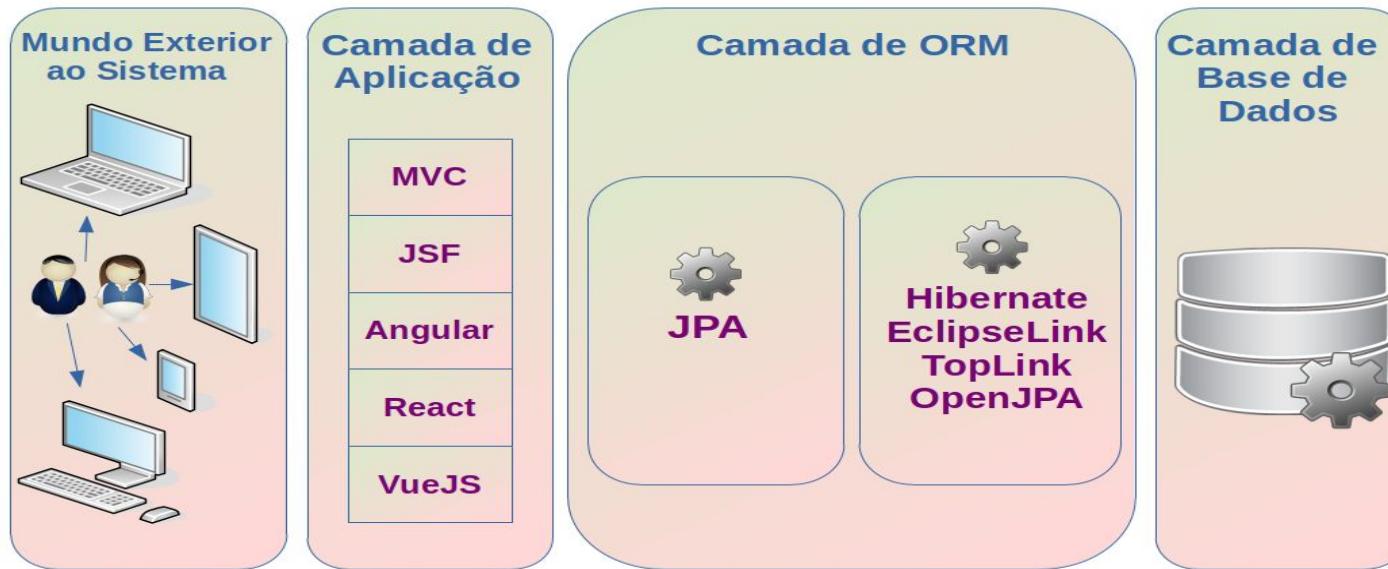
Abaixo segue um diagrama bem detalhado sobre ORM – Object Relational Mapping (Mapeamento Objeto Relacional)





JPA 2.0

No diagrama podemos ver o mundo exterior ao sistema onde os usuários podem acessar a aplicação de qualquer dispositivo, seja um computador, um tablet, um smartphone ou um notebook, eles vão através da aplicação chamar a camada ORM, onde se localiza a interface JPA e o framework utilizado seja o Hibernate, o EclipseLink, o TopLink, o OpenJPA, etc. E nesse diagrama podemos ver detalhadamente a divisão entre a interface JPA e o framework ORM utilizado.





JPA 2.0

Agora vou citar alguns objetos da API JPA:

EntityManagerFactory – Interface usada para interagir com a fábrica do gerenciador de entidades para a unidade de persistência.

Quando o aplicativo terminar de usar a fábrica do gerenciador de entidades e/ou no encerramento do aplicativo, o aplicativo deve fechar a fábrica do gerenciador de entidades. Depois que um EntityManagerFactory é fechado, todos os seus gerenciadores de entidade são considerados no estado fechado.



JPA 2.0

EntityManager – Interface usada para interagir com o contexto de persistência.

Uma instância de EntityManager está associada a um contexto de persistência. Um contexto de persistência é um conjunto de instâncias de entidade em que, para qualquer identidade de entidade persistente, existe uma instância de entidade única. Dentro do contexto de persistência, as instâncias de entidade e seu ciclo de vida são gerenciados. A API EntityManager é usada para criar e remover instâncias de entidade persistentes, para localizar entidades por sua chave primária e para consultar entidades.

O conjunto de entidades que podem ser gerenciadas por uma determinada instância EntityManager é definido por uma unidade de persistência. Uma unidade de persistência define o conjunto de todas as classes relacionadas ou agrupadas pelo aplicativo e que devem ser colocadas em seu mapeamento em um único banco de dados.



JPA Mapeamento

```
@Entity  
public class Book {  
  
    @Id  
    @GeneratedValue  
    private long id;  
  
    @Column(nullable=false)  
    private String title;  
  
    @ManyToOne  
    @JoinColumn(name="library_id")  
    private Library library;  
  
    // standard constructor, getter, setter  
}
```

```
public class Library {  
  
    //...  
  
    @OneToMany(mappedBy = "library")  
    private List<Book> books;  
  
    //...  
}
```



JPA Mapeamento

```
@Entity
public class Author {

    @Id
    @GeneratedValue
    private long id;

    @Column(nullable = false)
    private String name;

    @ManyToMany(cascade = CascadeType.ALL)
    @JoinTable(name = "book_author",
        joinColumns = @JoinColumn(name = "book_id", referencedColumnName = "id"),
        inverseJoinColumns = @JoinColumn(name = "author_id",
            referencedColumnName = "id"))
    private List<Book> books;

    //standard constructors, getters, setters
}
```

```
public class Book {

    //...

    @ManyToMany(mappedBy = "books")
    private List<Author> authors;

    //...
}
```



Entidades Managed, Transient e Detached no Hibernate e JPA

Distinguir entre os estados de uma entidade no JPA/Hibernate é difícil no início. Um objeto é dito transiente quando não tem representação no banco de dados e nem o EntityManager o conhece, como abaixo:

```
Cliente c = new Cliente();
```

Aqui, qualquer mudança no objeto referido por c não gerará nenhum tipo de insert ou update no banco de dados.



Entidades Managed, Transient e Detached no Hibernate e JPA

O oposto é quando o objeto existe no banco de dados e o EntityManager em questão possui uma referência para ele, essa entidade está *managed*, gerenciada pelo EntityManager. Considere em uma referência a um EntityManager no seguinte exemplo:

```
Cliente c = new Cliente(); // transiente  
em.persist(c); // gerenciado
```

Ou ainda:

```
Cliente c = em.find(Cliente.class, 1); // gerenciado
```

Quando uma entidade está managed, qualquer mudança em seu estado (como uma chamada de setter) resultará em uma atualização no banco de dados no momento do commit.



Entidades Managed, Transient e Detached no Hibernate e JPA

O último caso é quando a entidade representa algo que possivelmente está no banco de dados, mas o EntityManager o desconhece: a entidade está fora do contexto, *detached*.

Exemplo:

```
Cliente c = new Cliente();
c.setId(1);
```

Uma entidade também está *detached* quando o EntityManager de onde tiramos esse Cliente (por exemplo, quando fizemos um find ou vindo de uma Query) já não está mais aberta.

Qualquer mudança nessa referência obviamente não surtirá efeito no banco de dados. Para que essa mudança faça efeito, isto é, para *reattach* a entidade, antes precisamos amarrá-la ao contexto de persistência. Repare que no EntityManager já pode existir uma entidade Cliente com esse mesmo id, imagine então o que aconteceria se tivéssemos um método que se chamasse *reattach* ou *update*?



Entidades Managed, Transient e Detached no Hibernate e JPA

Por isso o método é o merge. Ele junta a possível entidade com mesmo id que se encontra no EntityManager com a passada como argumento, e devolve a que está *managed*. O método merge **não faz reattach**. Então:

```
Cliente c = new Cliente();
c.setId(1);
em.merge(c);
c.setNome("Cliente com nome alterado");
```

Não surtirá efeito! Aqui você precisava antes ter pego o que o merge devolveu. Repare na pequena alteração:

```
Cliente c = new Cliente();
c.setId(1);
c = em.merge(c);
c.setNome("Cliente com nome alterado");
```



JPA Ciclo de vida

```
@PrePersist  
public void logNewUserAttempt() {  
    log.info("Attempting to add new user with username: " + userName);  
}  
  
@PostPersist  
public void logNewUserAdded() {  
    log.info("Added user '" + userName + "' with ID: " + id);  
}  
  
@PreRemove  
public void logUserRemovalAttempt() {  
    log.info("Attempting to delete user: " + userName);  
}  
  
@PostRemove  
public void logUserRemoval() {  
    log.info("Deleted user: " + userName);  
}
```



JPA Ciclo de vida

```
@PreUpdate  
public void logUserUpdateAttempt() {  
    log.info("Attempting to update user: " + userName);  
}  
  
@PostUpdate  
public void logUserUpdate() {  
    log.info("Updated user: " + userName);  
}  
  
@PostLoad  
public void logUserLoad() {  
    fullName = firstName + " " + lastName;  
}
```



Encerramos por hoje !!!

