

Instituto Nacional de Telecomunicações – INATEL

**Pós-graduação em Desenvolvimento de Aplicações para
Dispositivos Móveis e Cloud Computing**

**DM107 - Desenvolvimento de Web Services com segurança sob
plataforma Java e PHP**

Parte 1

Sobre a apostila

Este é um material de apoio para o curso de Pós-graduação em Desenvolvimento de Aplicações para Dispositivos Móveis e *Cloud Computing* do Instituto Nacional de Telecomunicações – INATEL, referente a disciplina DM107 - Desenvolvimento de Web Services com segurança sob plataforma Java e PHP.

SUMÁRIO

| | | |
|----------|--|----|
| 1. | INTRODUÇÃO A REST WEB SERVICES | 8 |
| 1.1. | INTRODUÇÃO | 8 |
| 1.1.1. | Cliente servidor | 8 |
| 1.1.2. | <i>Stateless</i> | 8 |
| 1.1.3. | <i>Cache</i> | 9 |
| 1.1.4. | Interface Uniforme | 9 |
| 1.1.5. | Sistema em Camadas | 9 |
| 1.1.6. | Código sob Demanda | 9 |
| 1.2. | REST OU RESTFul | 9 |
| 1.3. | PROTOCOLO HTTP | 10 |
| 1.3.1. | Composição de uma <i>Uniform Resource Locator</i> (URL) HTTP ... | 10 |
| 1.3.2. | Recursos | 10 |
| 1.3.3. | Representações | 11 |
| 1.3.4. | Requisições e Respostas | 11 |
| 1.3.5. | Métodos | 13 |
| 1.3.5.1. | GET | 13 |
| 1.3.5.2. | POST | 13 |
| 1.3.5.3. | PUT | 13 |
| 1.3.5.4. | DELETE | 13 |
| 1.3.6. | Cabeçalhos | 14 |
| 1.3.7. | Media Types | 16 |
| 1.3.8. | Código de <i>status</i> | 17 |
| 2. | GRADLE | 21 |
| 2.1. | INTRODUÇÃO | 21 |

| | | |
|--------|---|----|
| 2.2. | EVOLUÇÃO DAS FERRAMENTAS DE <i>BUILD</i> JAVA..... | 22 |
| 2.3. | CARACTERÍSTICAS | 23 |
| 2.3.1. | Convenções Flexíveis..... | 25 |
| 2.3.2. | Robusta e Poderosa gestão de dependência | 25 |
| 2.3.3. | <i>Builds</i> Escaláveis..... | 26 |
| 2.3.4. | Extensibilidade sem esforço | 27 |
| 2.3.5. | Integração com outras ferramentas de <i>build</i> | 27 |
| 2.3.6. | Orientado pela comunidade e apoiado por empresas | 28 |
| 2.4. | <i>PLUGINS</i> GRADLE..... | 28 |
| 2.4.1. | Tipos de <i>Plugins</i> Gradle..... | 29 |
| 2.4.2. | Uso de <i>plugins</i> | 29 |
| 2.5. | INSTALANDO O GRADLE..... | 29 |
| 2.6. | OPÇÕES DE COMANDOS GRADLE..... | 32 |
| 2.7. | USANDO O SISTEMA DE COMPILAÇÃO GRADLE NO ECLIPSE IDE 33 | |
| 2.7.1. | Suporte do Eclipse Gradle..... | 33 |
| 2.7.2. | Instalando o Eclipse | 33 |
| 2.7.3. | Instalação do Gradle (Buildship) via Eclipse Marketplace..... | 34 |
| 2.8. | CRIANDO PROJETOS GRADLE..... | 34 |
| | REFERÊNCIAS | 55 |

LISTA DE FIGURAS

| | |
|--|----|
| Figura 1 - Formato geral de uma mensagem de requisição HTTP | 12 |
| Figura 2 - Formato geral de uma mensagem de resposta HTTP | 13 |
| Figura 3 - cabeçalho Requisição HTTP..... | 14 |
| Figura 4 - Gradle combina os melhores recursos de outras ferramentas de compilação | 23 |
| Figura 5- Comparação do tamanho do script de compilação e a legibilidade entre Maven e Gradle..... | 24 |
| Figura 6- Conjunto de características | 25 |
| Figura 7 - Instalação do Gradle | 30 |
| Figura 8 – Criando a variável de ambiente GRADLE_HOME no Windows..... | 31 |
| Figura 9 – Atualizando a variável de ambiente PATH no Windows | 31 |
| Figura 10 - Informação sobre a instalação do Gradle..... | 32 |
| Figura 11- Eclipse Marketplace | 34 |
| Figura 12 - Busca da ferramenta Buildship no Eclipse Marketplace | 34 |
| Figura 13 - Criação de um projeto Gradle | 35 |
| Figura 14 - Criação de um projeto Gradle | 35 |
| Figura 15 - Criação de um projeto Gradle | 36 |
| Figura 16 – Projeto Gradle | 36 |
| Figura 17 – Baixar as dependências em um projeto Gradle | 38 |
| Figura 18 - Executar Projeto | 39 |
| Figura 19 - Executar Projeto | 40 |
| Figura 20 - Resultado da execução do projeto..... | 40 |
| Figura 21 - Removendo arquivos na criação de um projeto Gradle | 41 |
| Figura 22 - Resultado da execução do projeto..... | 44 |
| Figura 23- Removendo arquivos na criação de um projeto Gradle | 45 |
| Figura 24 - Criando o <i>folder</i> webapp..... | 48 |

| | |
|---|----|
| Figura 25 - Criando o <i>folder</i> webapp..... | 48 |
| Figura 26 - Abrindo a aba Gradle Tasks..... | 50 |
| Figura 27 - Abrindo a aba Gradle Tasks..... | 50 |
| Figura 28- Run Gradle Tasks..... | 51 |
| Figura 29 - Run Gradle Tasks..... | 51 |
| Figura 30- Execução do Run Gradle Tasks | 52 |
| Figura 31 - Configuração para execução de um projeto Gradle Web | 52 |
| Figura 32 - Configuração para execução de um projeto Gradle Web | 53 |
| Figura 33 - Configuração para execução de um projeto Gradle Web | 53 |
| Figura 34 - Execução do projeto NovoAnoWeb | 54 |
| Figura 35 - Acessando o projeto NovoAnoWeb no browser..... | 54 |

LISTA DE QUADROS

| | |
|---|----|
| Quadro 1 - Alguns cabeçalhos de mensagens HTTP..... | 14 |
|---|----|

1. INTRODUÇÃO A REST WEB SERVICES

1.1. INTRODUÇÃO

REST surgiu no início dos anos 2000, a partir da tese do cientista chamado Roy Fielding.

O intuito geral era a formalização de um conjunto de melhores práticas denominadas *constraints*. Essas *constraints* tinham como objetivo determinar a forma na qual os padrões como o *Hypertext Transfer Protocol* (HTTP) e *Uniform Resource Identifier* (URI) deveriam ser modelados, aproveitando de fato todos os recursos oferecidos pelos mesmos.

1.1.1. Cliente servidor

A principal característica dessa *constraint* é separar as responsabilidades de diferentes partes de um sistema. Essa divisão pode se dar de diversas formas, iniciando por exemplo com uma separação entre mecanismos de armazenamento de dados e o *back-end* da aplicação.

Outra divisão é entre a interface do usuário e *back-end*. Isso permite a evolução e escalabilidade destas responsabilidades de forma independente. Atualmente várias ferramentas seguem este modelo, *frameworks* como AngularJS e *Application Programming Interface* (API) RESTful permitem o isolamento de forma elegante para cada uma dessas funcionalidades.

1.1.2. Stateless

Essa característica propõe que cada requisição ao servidor não deve ter ligação com requisições anteriores ou futuras, ou seja, cada requisição deve conter todas as informações necessárias para que ela seja tratada com sucesso pelo servidor.

O protocolo HTTP segue esse modelo, porém, é muito comum o uso de *cookies* para armazenamento de sessões do lado do servidor. Essa abordagem deve ser usada com cautela, visto os inconvenientes que a mesma carrega.

Um dos principais inconvenientes no uso de *cookies* é que sua utilização diminui de forma drástica a forma na qual pode-se escalar aplicações.

1.1.3. *Cache*

Para uma melhor performance, um sistema REST deve permitir que suas respostas sejam passíveis de *cache*. Cada resposta deve de alguma maneira informar para clientes ou elementos intermediários (*proxies*, *gateways* e/ou balanceadores de carga) qual a política de *cache* mais adequada.

O protocolo HTTP permite o funcionamento adequado dessa *constraint* a partir da adição dos *headers* “*expires*” (versão 1.0 do HTTP) e/ou “*cache-control*” (versão 1.1 do HTTP).

1.1.4. Interface Uniforme

Esta *constraint* é uma das mais importantes. Quando se fala sobre uma interface, os elementos abaixo devem ser considerados:

- Recursos;
- Mensagens auto descritivas;
- *Hypermedia*.

Quando implementada utilizando HTTP, uma API deve levar em consideração também a correta utilização dos métodos e códigos de retorno.

1.1.5. Sistema em Camadas

Com o intuito de permitir a escalabilidade necessária para grandes sistemas distribuídos, um sistema REST deve ter a capacidade de adicionar elementos intermediários e que sejam totalmente transparentes para seus clientes.

Tais elementos intermediários são utilizados de forma transparente para o cliente. Isso é algo de bastante valor, imagine acessar o site do Google sem um mecanismo de *Domain Name System* (DNS), ou se em cada acesso fosse necessário informar o desejo ou não de fazer um *cache*.

1.1.6. Código sob Demanda

Código sob demanda possibilita adaptar o cliente de acordo com novos requisitos e funcionalidades. Exemplos disso são o Java Script e Applets.

1.2. REST OU RESTful

Qual a diferença dos termos REST e RESTful? Mas de fato, uma API ou aplicação deve receber o nome REST ou RESTful?

O correto é dizer que uma API é RESTful. Além disso, é importante entender o porquê dessa nomenclatura e qual o momento certo de usar cada uma.

Quando se discute sobre o modelo e sobre as características (*constraints*), deve-se utilizar o termo REST, já no momento que se estiver falando de uma implementação que usa essas mesmas características, deve-se usar RESTful.

1.3. PROTOCOLO HTTP

1.3.1. Composição de uma *Uniform Resource Locator* (URL) HTTP

Uma URL HTTP é composta da seguinte maneira:

<protocol>://<host>:<port>/<path>?<searchpart>

Onde:

<protocol> é o nome do protocolo usado para acessar o documento. No contexto desta apostila, o protocolo em questão é o HTTP. A segunda informação é o <host>, que especifica o nome do domínio do servidor (hospedeiro) que abriga o documento. O item <port> informa o número da porta do protocolo. Essa informação pode ser omitida na URL, caso seja usada a porta 80, que é a porta padrão do HTTP. O item seguinte é o <path>, que informa o caminho do objeto no servidor. Esse caminho é percorrido a partir do diretório raiz, que é informado na configuração do servidor. Além disso, o caminho pode ser omitido quando o objetivo requisitado se encontra no diretório raiz e é o arquivo base da página Web, geralmente nomeado de index.html. Por fim, tem-se o <searchpart>, que é opcional. Este último item representa uma sequência de argumentos conhecidos como query strings.

1.3.2. Recursos

Um recurso é utilizado para identificar de forma única um objeto abstrato ou físico. A RFC 3986 especifica detalhadamente todas as características que uma URI deve seguir.

Basicamente, tem-se a modelagem de um recurso sob um substantivo, ao invés de verbos, como é usualmente utilizado em APIs. Exemplos:

/cliente/1

/produto/1

/cliente/1/notificacao

Uma URI deve ser visualizada como um recurso, e não como uma ação a ser executada sob um servidor. Alguns autores e literaturas defendem a utilização de verbos em casos específicos, porém, deve-se usar com cautela.

Quando um recurso é solicitado por um cliente (ex: *browser*), o servidor executa uma série de atividades e retorna uma mensagem ao solicitante. Essa mensagem é de fato uma representação de um determinado recurso. A mesma não necessariamente precisa estar ligada a alguma parte concreta de um sistema, como por exemplo, uma tabela de banco de dados.

1.3.3. Representações

As representações podem ser modeladas em vários formatos, como XML, JSON, HTML e etc. Apenas deve-se levar em consideração que esse formato deve suportar a utilização de *Hypermedia*.

Isso pode ser feito devido a capacidade de múltiplas representações (mais conhecido como negociação de conteúdo) que o HTTP possui. Para que isso seja possível, pode-se usar o *header* “*Accept*” para especificar qual é o tipo de representação mais adequada.

1.3.4. Requisições e Respostas

As mensagens HTTP podem ser de dois tipos: requisição ou resposta. Ambas seguem padrões estruturais definidos pelo protocolo. De forma genérica, a mensagem é composta por uma linha inicial, zero ou mais linhas de cabeçalho, uma linha em branco que indica o fim do cabeçalho, e por fim o corpo da mensagem, que é opcional a depender da situação.

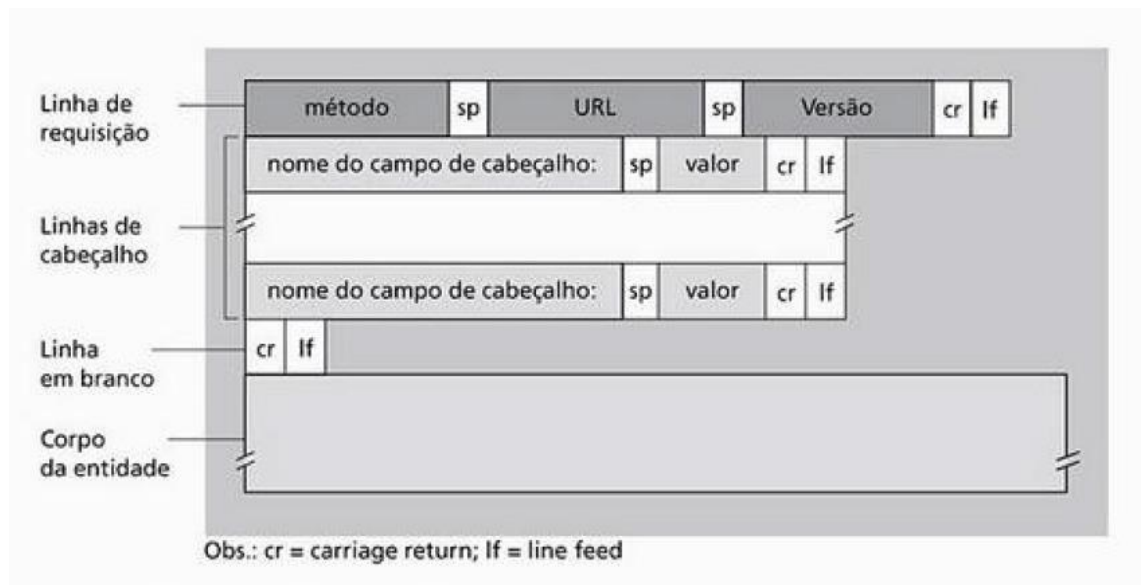


Figura 1 - Formato geral de uma mensagem de requisição HTTP

Fonte KUROSE e ROSS, 2010.

A primeira linha da mensagem é a linha de requisição, que indica o método utilizado, o objeto requisitado e a versão do protocolo. O método informa qual a ação a ser realizada no recurso especificado.

O objeto requisitado é a parte da URL que especifica o caminho do recurso no servidor e a versão do protocolo é a versão em uso do HTTP. Na sequência estão as linhas de cabeçalho, que possuem informações adicionais sobre a mensagem. Os campos cr e lf são caracteres especiais que representam *carriage return* e *line feed*, que fazem com que o cursor aponte para o início da próxima linha.

A mensagem de resposta que o servidor envia ao navegador segue o mesmo padrão, diferindo na linha inicial e em alguns cabeçalhos que são específicos da resposta. Como pode ser verificado na Figura 2, a primeira linha da mensagem de resposta do HTTP 1.1, conhecida como linha de estado, possui três campos: o campo da versão do protocolo, um código de estado e uma mensagem de estado correspondente.

O código de estado informa ao navegador se a solicitação foi atendida, ou seja, indica a resposta do servidor para a operação solicitada. Cada código de estado possui três dígitos. O primeiro dígito define a classe da resposta, podendo representar cinco categorias. Os dois últimos dígitos não possuem nenhuma regra de classificação.

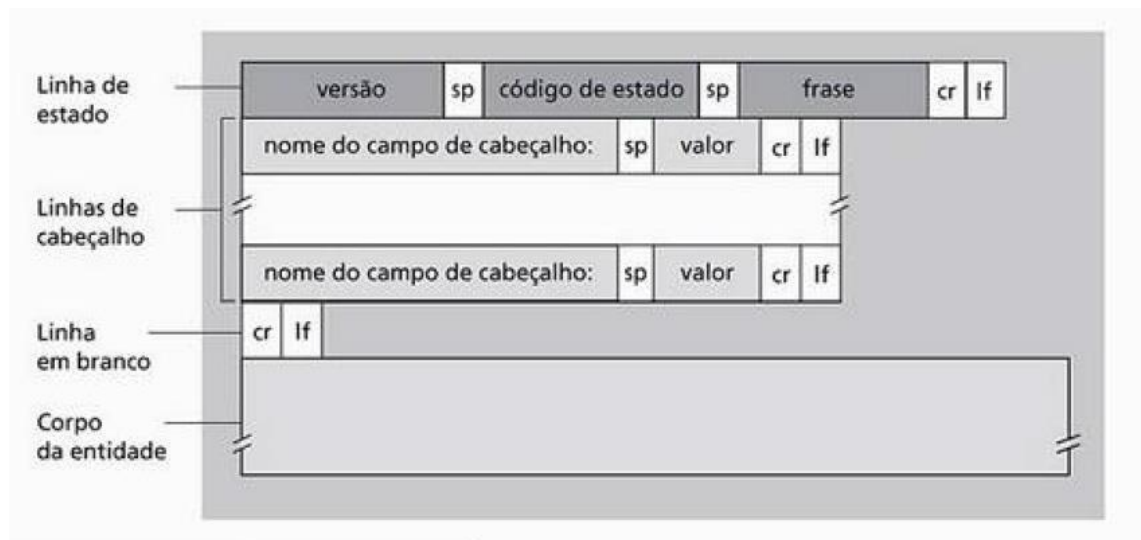


Figura 2 - Formato geral de uma mensagem de resposta HTTP

Fonte KUROSE e ROSS, 2010.

1.3.5. Métodos

A RFC 7231 especifica um conjunto de 8 métodos para a criação de uma API RESTful. Esse conjunto de métodos possui a semântica de operações possíveis de serem efetuadas sob um determinado recurso.

Dentre esses 8 métodos, abaixo segue um detalhamento dos 4 mais conhecidos.

1.3.5.1. GET

O método *GET* é utilizado quando existe a necessidade de se obter um recurso.

Ele é considerado idempotente, ou seja, independentemente da quantidade de vezes que é executado sob um recurso, o resultado sempre será o mesmo.

1.3.5.2. POST

Utilizado para a criação de um recurso a partir do uso de uma representação.

1.3.5.3. PUT

O método *PUT* é utilizado como forma de atualizar um determinado recurso. Em alguns cenários muito específicos, ele também pode ser utilizado como forma de criação, por exemplo quando o cliente tem a liberdade de decidir a URI a ser utilizada.

1.3.5.4. DELETE

O delete tem como finalidade a remoção de um determinado recurso.

1.3.6. Cabeçalhos

Os cabeçalhos, em HTTP, são utilizados para trafegar todo o tipo de meta informação a respeito das requisições. Vários destes cabeçalhos são padronizados, no entanto, eles são facilmente extensíveis para comportar qualquer particularidade que uma aplicação possa requerer nesse sentido.

Por exemplo, ao realizar uma requisição, pelo Firefox, para o site <http://www.inatel.br/home/> as seguintes informações são enviadas conforme Figura 3.

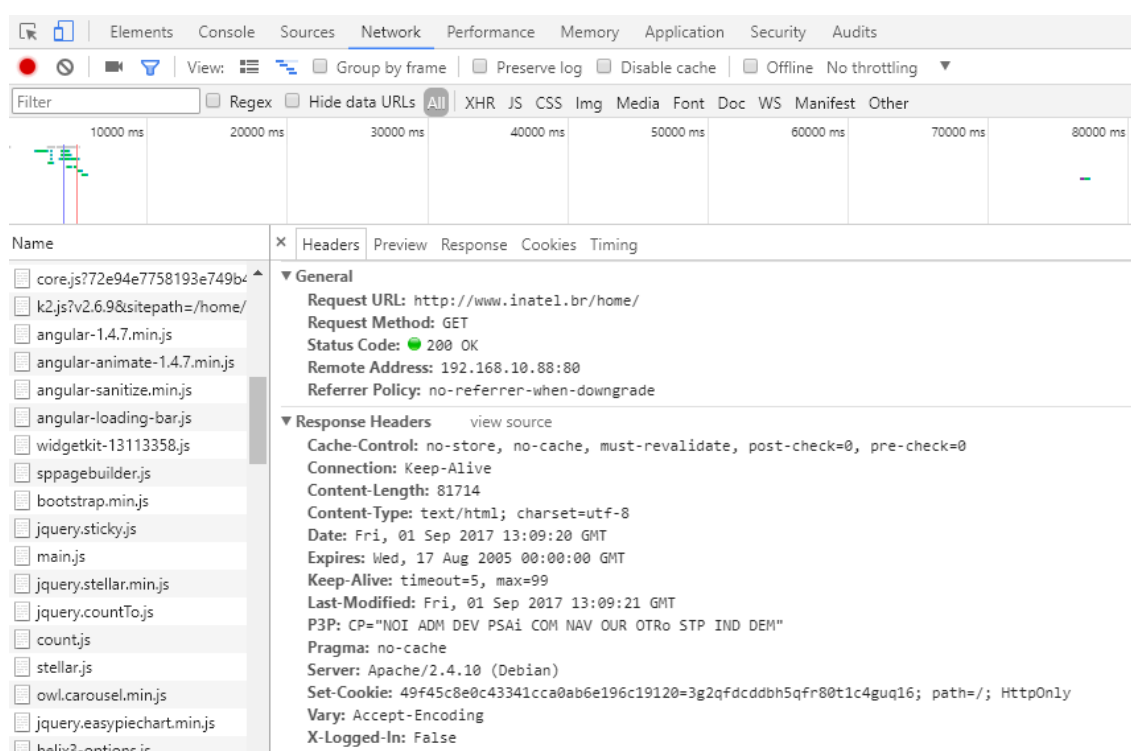


Figura 3 - cabeçalho Requisição HTTP

Para visualizar o tráfego destes dados, várias técnicas estão à disposição. As mais simples são a instalação de um *plugin* do Firefox chamado *Live HTTP Headers* ou apertar F12 no Chrome.

Cada um desses cabeçalhos tem um significado para o servidor, no entanto, o protocolo HTTP não exige nenhum deles. Ao estabelecer esta liberdade, tanto o cliente quanto o servidor estão livres para negociar o conteúdo da maneira como acharem melhor.

Obviamente, isto não quer dizer que estes cabeçalhos não são padronizados.

No quadro 1 são apresentados alguns cabeçalhos de mensagens HTTP.

| Cabeçalho | Tipo | Conteúdo |
|------------------|-------------|--|
| User-Agent | Solicitação | Informações sobre o navegador e sua plataforma |
| Accept | Solicitação | O tipo de páginas que o cliente pode manipular |
| Accept-Charset | Solicitação | Os conjuntos de caracteres aceitáveis para o cliente |
| Accept-Encoding | Solicitação | As codificações de páginas que o cliente pode manipular |
| Accept-Language | Solicitação | Os idiomas com os quais o cliente pode lidar |
| Host | Solicitação | O nome DNS do servidor |
| Authorization | Solicitação | Uma lista das credenciais do cliente |
| Cookie | Solicitação | Envia um cookie definido anteriormente de volta ao servidor |
| Date | Ambos | Data e hora em que a mensagem foi enviada |
| Upgrade | Ambos | O protocolo para o qual o transmissor deseja alternar |
| Server | Resposta | Informações sobre o servidor |
| Content-Encoding | Resposta | Como o conteúdo está codificado (por exemplo, gzip) |
| Content-Language | Resposta | O idioma usado na página |
| Content-Length | Resposta | O comprimento da página em bytes |
| Content-Type | Resposta | O tipo MIME da página |
| Last-Modified | Resposta | Data e hora da última modificação na página |
| Location | Resposta | Um comando para o cliente enviar sua solicitação a outro lugar |

| | | |
|---------------|----------|---|
| Accept-Ranges | Resposta | O servidor aceitará solicitações de intervalos de bytes |
| Set-Cookie | Resposta | O servidor deseja que o cliente grave um cookie |

1.3.7. Media Types

Ao realizar uma requisição para o site <http://www.inatel.br/home>, o *Media Type* `text/html` é utilizado, indicando para o navegador qual é o tipo da informação que está sendo trafegada (no caso, HTML). Isto é utilizado para que o cliente saiba como trabalhar com o resultado (e não com tentativa e erro, por exemplo), dessa maneira, os *Media Types* são formas padronizadas de descrever uma determinada informação.

Os *Media Types* são divididos em tipos e subtipos, e acrescidos de parâmetros (se houverem). São compostos com o seguinte formato: **tipo/subtipo**. Se houverem parâmetros, o `;` (ponto-e-vírgula) será utilizado para delimitar a área dos parâmetros.

Portanto, um exemplo de *Media Type* seria **`text/xml; charset="utf-8"`** (para descrever um XML cuja codificação seja UTF-8).

Os tipos mais comuns são:

- **application**
- **audio**
- **image**
- **text**
- **video**
- **vnd**

Cada um desses tipos é utilizado com diferentes propósitos. *Application* é utilizado para tráfego de dados específicos de certas aplicações. **Audio** é utilizado para formatos de áudio. **Image** é utilizado para formatos de imagens. **Text** é utilizado para formatos de texto padronizados ou facilmente inteligíveis por humanos. **Video** é utilizado para formatos de vídeo. **Vnd** para tráfego de informações de *softwares* específicos (por exemplo, o Microsoft Office).

Em serviços REST, vários tipos diferentes de *Media Types* são utilizados. As maneiras mais comuns de representar dados estruturados são via XML e JSON, que são representados pelos *Media Types* `application/xml` e `application/json`, respectivamente. OXML também pode ser representado por **text/xml**, desde que possa ser considerado legível por humanos.

Os *Media Types* são negociados a partir dos cabeçalhos *Accept* e *Content-Type*. O primeiro é utilizado em requisições, e o segundo, em respostas.

Ao utilizar o cabeçalho *Accept*, o cliente informa ao servidor qual tipo de dados espera receber. Caso seja o tipo de dados que não possa ser fornecido pelo servidor, o mesmo retorna o código de erro 415, indicando que o *Media Type* não é suportado. Se o tipo de dados existir, então os dados são fornecidos e o cabeçalho *Content-Type* apresenta qual é o tipo de informação fornecida.

Existem várias formas de realizar esta solicitação. Caso o cliente esteja disposto a receber mais de um tipo de dados, o cabeçalho *Accept* pode ser utilizado para esta negociação. Por exemplo, se o cliente puder receber qualquer tipo de dados, esta solicitação pode utilizar um curinga, representado por `*` (asterisco). O servidor irá converter esta solicitação em um tipo adequado.

Além disso, também é possível solicitar mais de um tipo de dados apenas separando-os por `,` (virgula).

1.3.8. Código de *status*

Toda requisição enviada para o servidor retorna um código de status. Esses códigos são divididos em cinco famílias: 1xx, 2xx, 3xx, 4xx e 5xx, sendo:

- 1xx - Informativos
- 2xx - Códigos de sucesso
- 3xx - Códigos de redirecionamento
- 4xx - Erros causados pelo cliente
- 5xx - Erros originados no servidor

De acordo com Leonard Richardson e Sam Ruby, os códigos mais importantes e mais utilizados são:

2xx

200 - OK

Indica que a operação indicada teve sucesso.

201 - Created

Indica que o recurso desejado foi criado com sucesso. Deve retornar um cabeçalho *Location*, que deve conter a URL onde o recurso recém-criado está disponível.

202 - Accepted

Indica que a solicitação foi recebida e será processada em outro momento. É tipicamente utilizada em requisições assíncronas, que não serão processadas em tempo real. Por esse motivo, pode retornar um cabeçalho *Location*, que trará uma URL onde o cliente pode consultar se o recurso já está disponível ou não.

204 - No Content

Usualmente enviado em resposta a uma requisição PUT, POST ou DELETE, onde o servidor pode recusar-se a enviar conteúdo.

206 - Partial Content

Utilizado em requisições GET parciais, ou seja, que demandam apenas parte do conteúdo armazenado no servidor (caso muito utilizado em servidores de *download*).

3xx

301 - Moved Permanently

Significa que o recurso solicitado foi realocado permanentemente. Uma resposta com o código 301 deve conter um cabeçalho *Location* com a URL completa (ou seja, com descrição de protocolo e servidor) de onde o recurso está atualmente.

303 - See Other

É utilizado quando a requisição foi processada, mas o servidor não deseja enviar o resultado do processamento. Ao invés disso, o servidor envia a resposta com este código de status e o cabeçalho *Location*, informando onde a resposta do processamento está.

304 - Not Modified

É utilizado, principalmente, em requisições GET condicionais - quando o cliente deseja ver a resposta apenas se ela tiver sido alterada em relação a uma requisição anterior.

307 - *Temporary Redirect*

Similar ao 301, mas indica que o redirecionamento é temporário, não permanente.

4xx

400 - *Bad Request*

É uma resposta genérica para qualquer tipo de erro de processamento cuja responsabilidade é do cliente do serviço.

401 - *Unauthorized*

Utilizado quando o cliente está tentando realizar uma operação sem ter fornecido dados de autenticação (ou a autenticação fornecida for inválida).

403 - *Forbidden*

Utilizado quando o cliente está tentando realizar uma operação sem ter a devida autorização.

404 - *Not Found*

Utilizado quando o recurso solicitado não existe.

405 - *Method Not Allowed*

Utilizado quando o método HTTP utilizado não é suportado pela URL. Deve incluir um cabeçalho *Allow* na resposta, contendo a listagem dos métodos suportados (separados por “,”).

409 - *Conflict*

Utilizado quando há conflitos entre dois recursos. Comumente utilizado em resposta a criações de conteúdos que tenham restrições de dados únicos - por exemplo, criação de um usuário no sistema utilizando um *login* já existente. Se for causado pela existência de outro recurso (como no caso citado), a resposta deve conter um cabeçalho *Location*, explicitando a localização do recurso que é a fonte do conflito.

410 - *Gone*

Semelhante ao 404, mas indica que um recurso já existiu neste local.

412 - *Precondition failed*

Comumente utilizado em resposta a requisições GET condicionais.

415 - *Unsupported MediaType*

Utilizado em resposta a clientes que solicitam um tipo de dados que não é suportado - por exemplo, solicitar JSON quando o único formato de dados suportado é XML.

5xx

500 - *Internal Server Error*

É uma resposta de erro genérica, utilizada quando nenhuma outra se aplica.

503 - *Service Unavailable*

Indica que o servidor está atendendo requisições, mas o serviço em questão não está funcionando corretamente. Pode incluir um cabeçalho *Retry-After*, dizendo ao cliente quando ele deveria tentar submeter a requisição novamente.

2. GRADLE

2.1. INTRODUÇÃO

Durante anos, as compilações tinham requisitos simples de compilação e empacotamento de *software*. Mas o cenário do desenvolvimento de *software* moderno mudou, e assim tem-se novas necessidades para a automação de compilação. Hoje, os projetos envolvem pilhas de *software* grandes e diversas, incorporam múltiplas linguagens de programação, e aplicam um amplo espectro de estratégias de teste. Com o aumento das práticas ágeis, as *builds* têm que suportar a integração precoce do código, bem como possibilitar uma entrega fácil de testes para ambientes de produção.

O Gradle é o próximo passo evolutivo nas ferramentas de compilação baseadas em JVM. Baseia-se em lições aprendidas com ferramentas já estabelecidas como Ant e Maven e leva suas melhores práticas para o próximo nível. Seguindo uma abordagem de compilação por convenção, o Gradle permite modelar declarativamente seu domínio de problemas usando uma poderosa e expressiva linguagem específica de domínio (DSL) implementada em Groovy em vez de XML. Como a ferramenta Gradle é nativa da JVM, permite que você escreva lógica personalizada no idioma com o qual você se sente mais confortável, seja Java ou Groovy.

No mundo de Java, um número incrivelmente grande de bibliotecas e *frameworks* estão disponíveis. O gerenciamento de dependências é usado para baixar automaticamente esses artefatos de um repositório e torná-los disponíveis para o seu código. Tendo aprendido com as deficiências das soluções de gerenciamento de dependência existentes, a ferramenta Gradle fornece sua própria implementação. Não só é altamente configurável, mas também se esforça para ser o mais compatível possível com as infra-estruturas de gerenciamento de dependências existentes (como Maven e Ivy). A capacidade de Gradle para gerenciar dependências não se limita a bibliotecas externas. À medida um projeto cresce em tamanho e complexidade e deseja-se organizar o código em módulos com responsabilidades claramente definidas. O Gradle oferece suporte poderoso para definir e organizar compilações de multiprojetos, além de modelar dependências entre projetos.

2.2. EVOLUÇÃO DAS FERRAMENTAS DE *BUILD* JAVA

Duas ferramentas dominaram a construção de projetos Java: Ant e Maven. Ao longo dos anos, ambas as ferramentas melhoraram significativamente e ampliaram seu conjunto de recursos. Mas mesmo que ambas sejam altamente populares e se tornaram padrões da indústria, elas têm um ponto fraco: a lógica de compilação precisa ser descrita em XML. O XML é ótimo para descrever dados hierárquicos, mas é insuficiente para expressar o fluxo do programa e a lógica condicional. À medida que um *script* de compilação cresce em complexidade, manter o código de compilação torna-se um pesadelo.

A primeira versão oficial da ferramenta Ant foi lançada em 2000. Cada elemento de trabalho (um alvo Ant) pode ser combinado e reutilizado. Vários alvos podem ser encadeados para combinar unidades de trabalho únicas em fluxos de trabalho completos. Por exemplo, você pode ter um alvo para compilar o código-fonte Java e outro para criar um arquivo JAR que empacote os arquivos de classe.

Em Ant, você faz com que o alvo JAR dependa do alvo de compilação. Ant não fornece nenhuma orientação sobre como estruturar seu projeto. Embora permita uma flexibilidade máxima, Ant torna cada *script* de compilação único e difícil de entender. As bibliotecas externas exigidas pelo seu projeto geralmente foram verificadas no controle de versão, porque não havia nenhum mecanismo sofisticado para puxá-los automaticamente de uma localização central. As primeiras versões de Ant exigiram muita disciplina para evitar o código repetitivo.

A ferramenta Maven, foi lançada em julho de 2004, tentou facilitar esse processo. Forneceu um projeto padronizado e estrutura de diretório, bem como gerenciamento de dependências. Porém infelizmente, a lógica personalizada ainda era difícil de implementar. Se você quiser sair das convenções de Maven, escrever um plugin, chamado Mojo, geralmente é a única solução. Na realidade, escrever um plugin em Maven é excessivamente complexo.

Mais tarde, a Ant alcançou o Maven ao introduzir o gerenciamento de dependências através da biblioteca Apache Ivy, que pode ser totalmente integrada com a Ant para especificar declarativamente as dependências necessárias para o processo de compilação e empacotamento de seu projeto. A ferramenta Maven, bem como Ivy,

suportam a resolução de dependências transitivas. Quando falo de dependências transitivas, quero dizer o gráfico das bibliotecas exigido pelas dependências especificadas. Um exemplo típico de uma dependência transitiva seria a biblioteca XML Parser Xerces que exige que a biblioteca XML API funcione corretamente.

O Maven 2, lançado em outubro de 2005, levou ainda mais a idéia de convenção sobre a configuração. Os projetos que consistem em múltiplos módulos podem definir suas dependências entre si.

2.3. CARACTERÍSTICAS

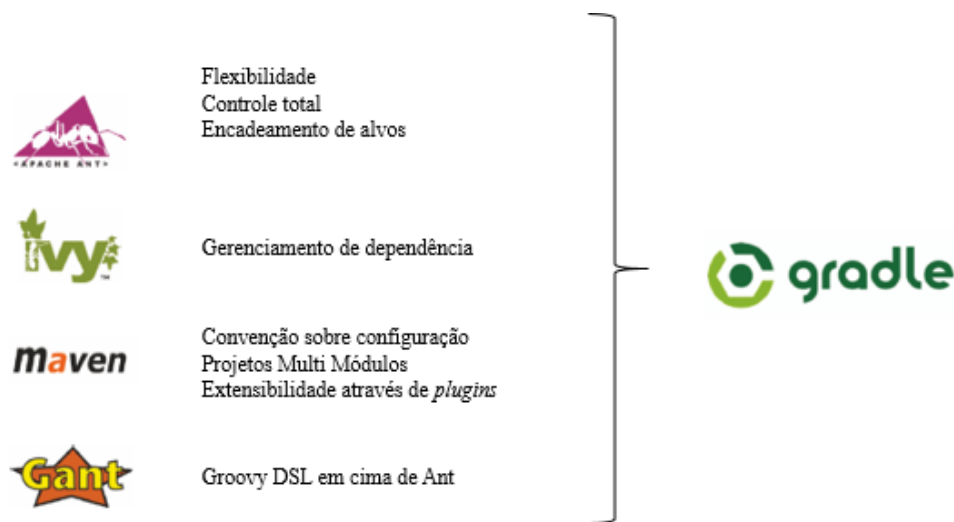


Figura 4 - Gradle combina os melhores recursos de outras ferramentas de compilação

O Gradle satisfaz muitos requisitos de ferramentas de compilação modernas conforme pode ser visto na Figura 4. Fornece uma DSL expressiva, uma abordagem de convenção sobre configuração e um poderoso gerenciamento de dependências. Ele faz um movimento para abandonar XML e introduzir o idioma dinâmico Groovy para definir a lógica de compilação.

Os *scripts* de compilação do Gradle são declarativos, legíveis e expressam claramente sua intenção. Escrever código em Groovy em vez de XML, reduz significativamente o tamanho de um *script* de compilação como pode ser visto na Figura 5.

Maven

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Gradle

```
apply plugin: 'java'
group = 'com.mycompany.app'
archivesBaseName = 'my-app'
version = '1.0-SNAPSHOT'

repositories {
    mavenCentral()
}

dependencies {
    testCompile 'junit:junit:4.11'
}
```

Figura 5- Comparação do tamanho do script de compilação e a legibilidade entre Maven e Gradle

Projetos populares de código aberto como Groovy e Hibernate mudaram completamente suas construções para Gradle. Todo projeto Android vem com o Gradle como o sistema de compilação padrão. O Gradle também teve impacto no mercado comercial. Companhias como Orbitz, EADS e *Software AG* também abraçaram Gradle, para citar apenas alguns. A VMware, a empresa atrás da Spring e Grails, fez investimentos significativos na escolha do Gradle. Muitos dos seus produtos de *software*, como o *framework* Spring e Grails, são construídos literalmente com base em Gradle.

Em resumo conforme Figura 6, Gradle é um sistema de compilação, alimentado por uma DSL Groovy declarativa e expressiva. Combina flexibilidade e extensibilidade sem esforço com a idéia de convenção sobre configuração e suporte para o gerenciamento tradicional de dependências.

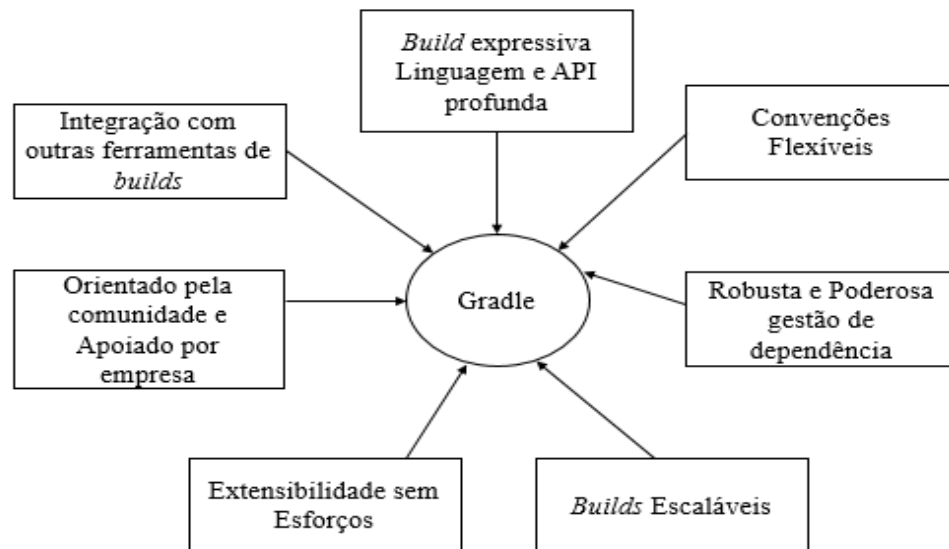


Figura 6- Conjunto de características

2.3.1. Convenções Flexíveis

Uma das grandes ideias da Gradle é dar-lhe diretrizes e padrões sensíveis para projetos.

Todo o projeto Java em Gradle sabe exatamente onde o arquivo de origem e a classe de teste devem estar, e como compilar seu código, executar testes de unidade, gerar relatórios Javadoc e criar uma distribuição de seu código. Todas estas tarefas estão totalmente integradas no ciclo de vida da compilação. Se a convenção for cumprida, há apenas um esforço de configuração mínimo.

2.3.2. Robusta e Poderosa gestão de dependência

Os projetos de *software* geralmente não são autônomos. Muitas vezes, o código usa uma biblioteca de terceiros que fornece uma funcionalidade existente para resolver um problema específico. Por que reinventar a roda implementando uma estrutura de persistência se existe o Hibernate? Dentro de uma organização, é possível ser consumidor de um componente ou módulo implementado por uma equipe diferente. Dependências externas são acessíveis através de repositórios, e o tipo de repositório é altamente dependente do que a empresa prefere. As opções variam de um sistema de arquivos simples para um repositório corporativo. As dependências externas podem ter uma referência a outras bibliotecas ou recursos. Chamamos essas dependências transitivas.

O Gradle fornece uma infra-estrutura para gerenciar a complexidade da resolução, recuperação e armazenamento de dependências. Uma vez que eles são baixados e

colocados no cache local, elas estão disponíveis para o projeto.

Um requisito fundamental para as construções empresariais é a reprodutibilidade. Construir um projeto deve produzir o mesmo resultado em máquinas diferentes, independentemente do conteúdo do cache local. Os gerentes de dependência como Ivy e Maven em sua implementação atual não garantem totalmente a reprodutibilidade.

Sempre que uma dependência é baixada e armazenada no cache local, ela não leva em consideração a origem do artefato. Em situações em que o repositório é alterado para um projeto, a dependência em cache é considerada resolvida, mesmo que o conteúdo do artefato possa ser ligeiramente diferente. Na pior das hipóteses, isso causará uma compilação com falha que é extremamente difícil de depurar.

Outra queixa comum específica para a Ivy é o fato de que as versões instantâneas de dependência, os artefatos atualmente em desenvolvimento com a convenção de nomenclatura -SNAPSHOT, não são atualizados corretamente no cache local, mesmo que ele tenha sido alterado no repositório e está marcado como alterado. Existem muitos outros cenários em que as soluções atuais ficam aquém.

Projetos de grandes empresas geralmente consistem em múltiplos módulos para separar a funcionalidade. No mundo Gradle, cada um dos submódulos é considerado um projeto que pode definir dependências para bibliotecas externas ou outros módulos. Além disso, cada subprojeto pode ser executado individualmente. O Gradle descobre quais das dependências do subprojeto precisam ser reconstruídas, sem ter que armazenar o artefato de um subprojeto no cache local.

2.3.3. Builds Escaláveis

O Gradle suporta compilações incrementais especificando entradas e saídas de tarefas. Ele calcula de forma confiável quais tarefas precisam ser ignoradas, construídas ou parcialmente reconstruídas. O mesmo conceito se traduz em projetos multimodulares, chamados de compilações parciais. Como a compilação define claramente as dependências entre os submódulos, o Gradle cuida da reconstrução apenas das peças necessárias.

A unidade automatizada, a integração e os testes funcionais fazem parte do processo de construção. Faz sentido separar os tipos de testes curtos dos que exigem a

configuração de recursos ou dependências externas para serem executados. O Gradle suporta a execução de testes paralelos. Esse recurso é totalmente configurável e garante que você realmente esteja aproveitando os núcleos do seu processador.

Os desenvolvedores executam compilações muitas vezes durante o desenvolvimento. Isso significa iniciar um novo processo Gradle de cada vez, carregar todas as suas dependências internas e executar a lógica de compilação. Pode-se notar que geralmente leva alguns segundos antes que o *script* realmente comece a ser executado. Para melhorar o desempenho de inicialização, o Gradle pode ser executado no modo *daemon*. Na prática, o comando Gradle garante um processo *daemon*, que não só executa sua compilação, mas também continua funcionando em segundo plano. As invocações de compilação subseqüentes serão encaminhadas para o processo do *daemon* existente para evitar os custos de inicialização. Como resultado, nota-se uma execução de compilação inicial muito mais rápida.

2.3.4. Extensibilidade sem esforço

A maioria das construções empresariais não são iguais, nem resolvem os mesmos problemas.

A maneira mais fácil de implementar a lógica personalizada é escrever uma *task*. As *tasks* podem ser definidas diretamente no *script* de compilação sem uma cerimônia especial. Se a complexidade aumentou, pode-se explorar a opção de criar uma *task* personalizada que permite escrever sua lógica dentro de uma definição de classe, tornando a estruturação do seu código fácil e sustentável. Se é necessário compartilhar o código reutilizável entre compilações e projetos, os *plugins* são a melhor forma. Representando o mecanismo de extensão mais poderoso do Gradle, os *plugins* oferecem acesso completo à API do Gradle e podem ser escritos, testados e distribuídos como qualquer outro *software*.

2.3.5. Integração com outras ferramentas de *build*

O Gradle trabalha bem com seus antecessores Ant, Maven e Ivy. Se você vem do Ant, o Gradle não o obriga a migrar completamente sua infraestrutura de construção. Em vez disso, ele permite que importe-se a lógica de compilação existente e reutilize tarefas Ant padrão.

As compilações do Gradle são 100% compatíveis com os repositórios Maven e Ivy. Pode-se recuperar dependências e publicar seus próprios artefatos. O Gradle fornece

um conversor para construções Maven existentes que podem traduzir a lógica de compilação em um *script* de compilação Gradle.

2.3.6. Orientado pela comunidade e apoiado por empresas

Gradle é gratuito e é fornecido com a Licença Apache 2.0.

Ao longo dos últimos cinco anos, os desenvolvedores de código aberto fizeram grandes contribuições para a base de código principal do Gradle. Ser hospedado no GitHub acabou por ser muito benéfico para o Gradle. As alterações de código podem ser submetidos e passar por um processo de revisão por parte dos principais *committers* antes de passar a fazer parte da base do código. Além dos *plugins* padrão fornecidos com o tempo de execução, a comunidade de Gradle libera novas funcionalidades quase que diariamente.

2.4. PLUGINS GRADLE

O Gradle no seu núcleo fornece intencionalmente muito pouco para a automação do mundo real. Todos os recursos úteis, como a capacidade de compilar código Java, são adicionados através de *plugins*. Os complementos adicionam novas tarefas (por exemplo, `JavaCompile`), objetos de domínio (por exemplo, `SourceSet`), convenções (por exemplo, a fonte Java está localizada em `src/main/java`), bem como a extensão de objetos e objetos do núcleo de outros *plugins*.

A aplicação de um *plugin* permite ampliar os recursos do projeto como:

- Estender o modelo Gradle (por exemplo, adicione novos elementos DSL que possam ser configurados);
- Configurar o projeto de acordo com as convenções (por exemplo, adicione novas tarefas ou configure valores padrões);
- Aplicar configuração específica (por exemplo, adicionar repositórios organizacionais ou aplicar padrões).

Ao aplicar *plugins*, ao invés de adicionar lógica ao *script* de construção do projeto, é possível colher uma série de benefícios:

- Promove a reutilização e reduz a sobrecarga da manutenção de lógica semelhante em vários projetos;
- Permite um maior grau de modularização, aprimorando a compreensão e organização;

- Encapsula a lógica imperativa e permite que os *scripts* de compilação sejam tão declarativos quanto possível;

2.4.1. Tipos de *Plugins* Gradle

Existem dois tipos gerais de *plugins* no Gradle, *plugins* de *script* e *plugins* binários.

Os *plugins* de *script* são *scripts* de compilação adicionais que geralmente implementam uma abordagem declarativa para manipular a compilação. Eles geralmente são usados dentro de uma compilação, embora possam ser externalizados e acessados a partir de um local remoto.

Os *plugins* binários são classes que implementam a interface `Plugin` e adotam uma abordagem programática para manipular a compilação. Os *plugins* binários podem residir dentro de um *script* de compilação, dentro da hierarquia do projeto ou externamente em um jar de *plugin*.

2.4.2. Uso de *plugins*

Para usar a lógica de compilação encapsulada em um *plugin*, o Gradle precisa executar duas etapas. Primeiro, ele precisa resolver o *plugin* e, em seguida, ele precisa aplicar o *plugin* ao alvo, geralmente um projeto.

Resolver um *plugin* significa encontrar a versão correta do jar que contém um determinado *plugin* e adicioná-lo ao *classpath*. Uma vez que um *plugin* é resolvido, sua API pode ser usada em um *script* de compilação. Os *plugins* de *script* são auto-resolvidos na medida em que são resolvidos a partir do caminho específico do arquivo ou do URL fornecido ao aplicá-los. Os *plugins* binários do núcleo fornecidos como parte da distribuição do Gradle são resolvidos automaticamente.

A aplicação de um *plugin* significa realmente executar o `Plugin.apply` do *plugin* (T) no Projeto que deseja aprimorar com o *plugin*. Pode-se aplicar com segurança qualquer *plugin* várias vezes sem efeitos colaterais.

2.5. INSTALANDO O GRADLE

Como pré-requisito, certifique-se de que já instalou o JDK com uma versão 1.7 ou superior.

Para verificar a versão JDK, execute o seguinte comando:

```
$ java -version
```

Posteriormente faça download da versão 4.1 no endereço <https://gradle.org/releases/> conforme Figura 7:

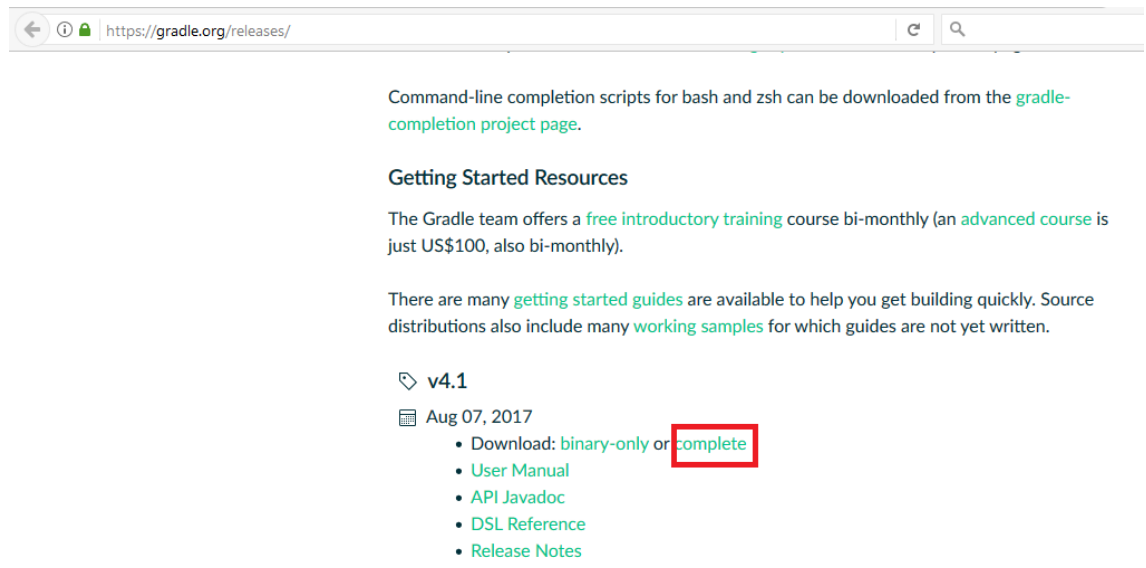


Figura 7 - Instalação do Gradle

Descompacte o arquivo baixado para um diretório de sua escolha. Para fazer referência em tempo de execução do Gradle no shell, você precisará criar a variável de ambiente `GRADLE_HOME` e adicionar os binários ao caminho de execução do seu shell:

- Mac OS X e * nix: para tornar o Gradle disponível no seu shell, adicione as duas linhas a seguir ao seu *script* de inicialização (por exemplo, `~ / .profile`). Essas instruções assumem que você instalou o Gradle no diretório `/opt/gradle`:

```
export GRADLE_HOME=/opt/gradle
export PATH=$PATH:$GRADLE_HOME/bin
```

- Windows: na janela de variável de ambiente, defina a variável `GRADLE_HOME`, verifique se a variável `JAVA_HOME` está criada e atualize as configurações do seu *path* conforme Figura 5 e 6:

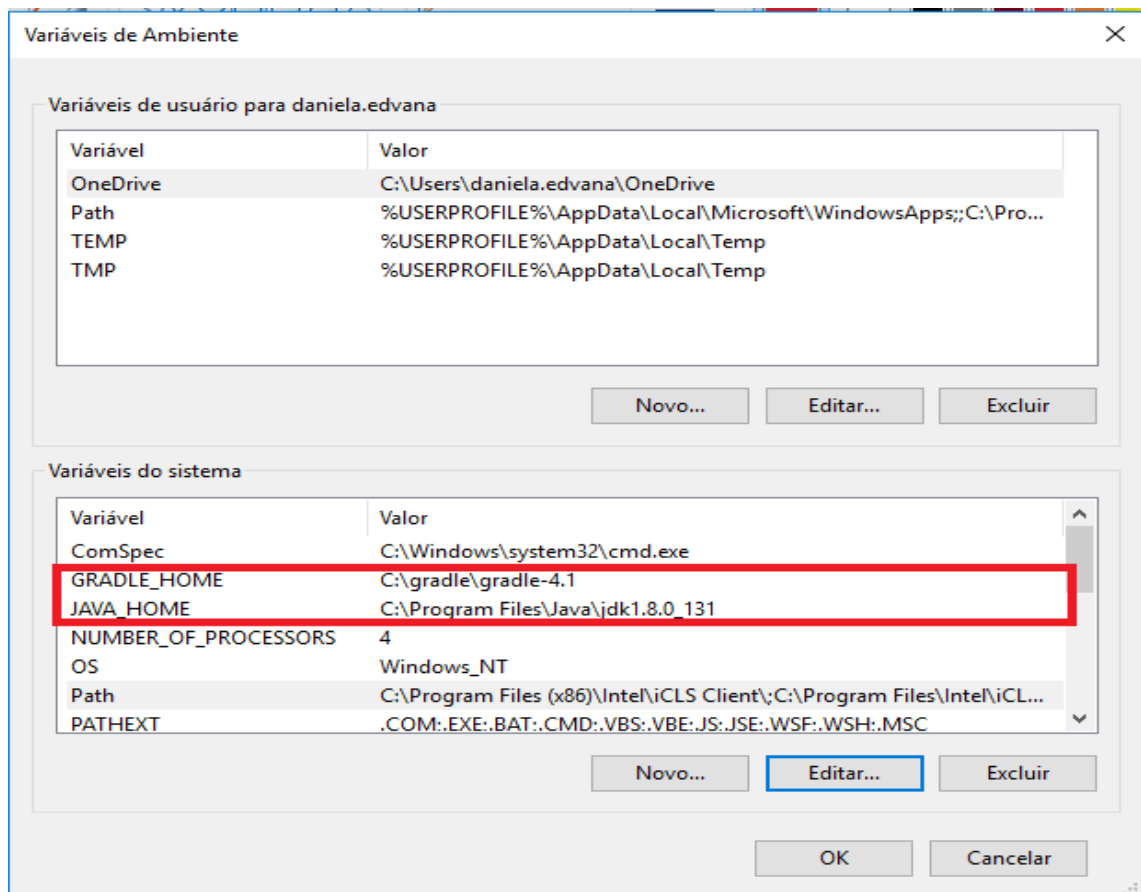


Figura 8 – Criando a variável de ambiente GRADLE_HOME no Windows

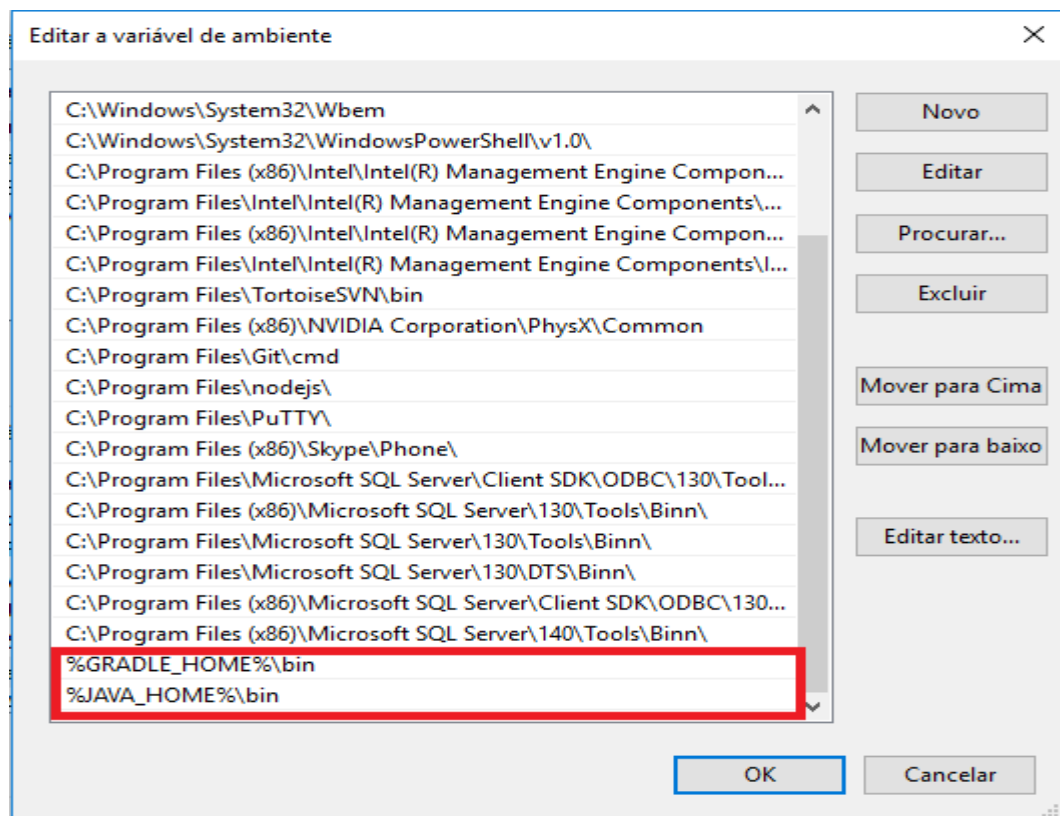
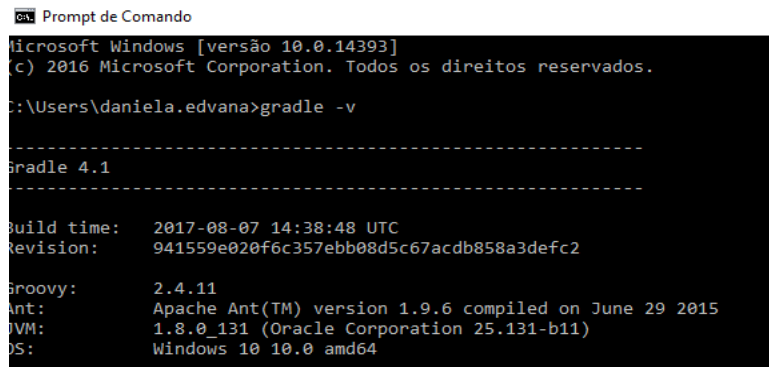


Figura 9 – Atualizando a variável de ambiente PATH no Windows

Para verificar se o Gradle foi instalado corretamente execute o comando:

```
$ gradle -v
```

Você deve ver meta-informações sobre a instalação, sua JVM conforme Figura 7.



```
Prompt de Comando
Microsoft Windows [versão 10.0.14393]
(c) 2016 Microsoft Corporation. Todos os direitos reservados.

C:\Users\daniela.edvana>gradle -v

-----
Gradle 4.1
-----

Build time:   2017-08-07 14:38:48 UTC
Revision:    941559e020f6c357ebb08d5c67acdb858a3defc2

Groovy:       2.4.11
Ant:          Apache Ant(TM) version 1.9.6 compiled on June 29 2015
JVM:          1.8.0_131 (Oracle Corporation 25.131-b11)
OS:           Windows 10 10.0 amd64
```

Figura 10 - Informação sobre a instalação do Gradle

Cada compilação Gradle começa com um *script*. A convenção de nomenclatura padrão para um *script* de compilação do Gradle é build.gradle. Ao executar o comando gradle em um shell, Gradle procura um arquivo com esse nome exato. Se não puder ser localizado, em tempo de execução será exibida uma mensagem de ajuda.

2.6. OPÇÕES DE COMANDOS GRADLE

- -?, -h, --help: imprime todas as opções de linha de comando disponíveis, incluindo uma mensagem descritiva.
- -b, --build-file: a convenção de nomenclatura padrão para o *script* de compilação do Gradle é build.gradle. Opção para executar um *script* de compilação com um nome diferente (por exemplo, gradle -b test.gradle).
- - *offline*: muitas vezes sua compilação declara dependências em bibliotecas somente disponíveis em repositórios fora da rede. Se essas dependências ainda não foram armazenadas no cache local, executar uma compilação sem conexão de rede a esses repositórios resultaria em uma compilação com falha. Esta opção é usada para executar sua compilação no modo *offline*.

Opções de Log

- `-i, --info`: nas configurações padrão, uma compilação Gradle não exibe muita informação. Esta opção é usada para obter mais mensagens informativas, alterando o *logger* Gradle para o nível de log INFO.
- `-s, --stacktrace`: A opção `-s` imprime um rastreamento abreviado da *stack trace* se uma exceção for lançada, tornando-o perfeito para depurar compilações quebradas.
- `-q, --quiet`: reduz as mensagens de log de uma execução de compilação apenas para mensagens de erro

Ajuda

- *tasks*: exibe todas as tarefas executáveis do seu projeto, incluindo suas descrições.

Os *plugins* aplicados ao seu projeto podem fornecer tarefas adicionais.

- *properties*: emite uma lista de todas as propriedades disponíveis no projeto. Algumas dessas propriedades são fornecidas pelo objeto do projeto de Gradle, a representação interna da compilação. Outras propriedades são propriedades definidas pelo usuário provenientes de uma opção de linha de comando de propriedade ou propriedade, ou diretamente declaradas em seu *script* de compilação.

2.7. USANDO O SISTEMA DE COMPILAÇÃO GRADLE NO ECLIPSE IDE

2.7.1. Suporte do Eclipse Gradle

A empresa Gradle Inc., responsável pela estrutura de compilação da Gradle fornece ferramentas Gradle para o Eclipse IDE. Esta ferramenta permite criar e importar projetos habilitados para Gradle para o Eclipse IDE. Também permite executar tarefas do Gradle e monitorá-la.

2.7.2. Instalando o Eclipse

Faça o download da versão mais atual aqui:

<http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/oxygenr>

O Eclipse roda direto de um arquivo executável. Extraia o pacote “**.zip**” do Eclipse para o diretório de sua preferência. Execute o arquivo `eclipse.exe` para iniciar.

Ao iniciar o eclipse, você deve definir o *workspace* de trabalho (pasta onde será mantido seu espaço de trabalho).

2.7.3. Instalação do Gradle (Buildship) via Eclipse Marketplace

A maneira mais fácil de instalar a ferramenta Gradle no Eclipse é usando o cliente Marketplace. Procure por Buildship conforme Figura 8 e 9.

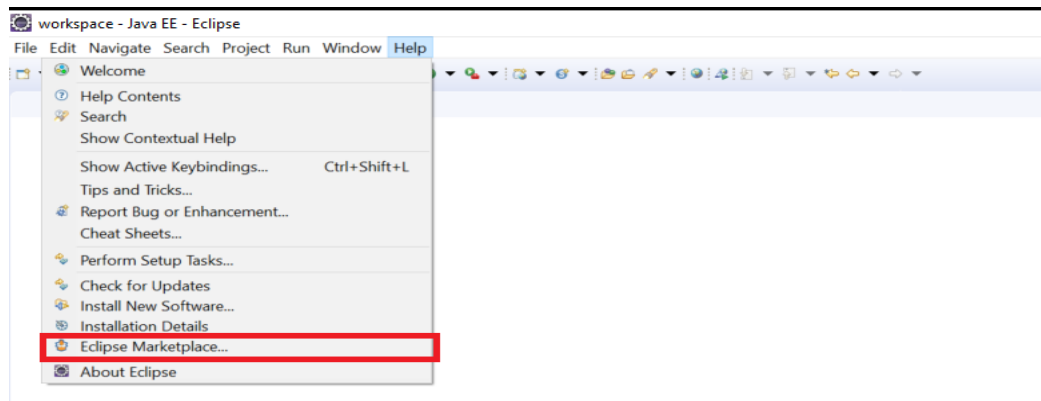


Figura 11- Eclipse Marketplace

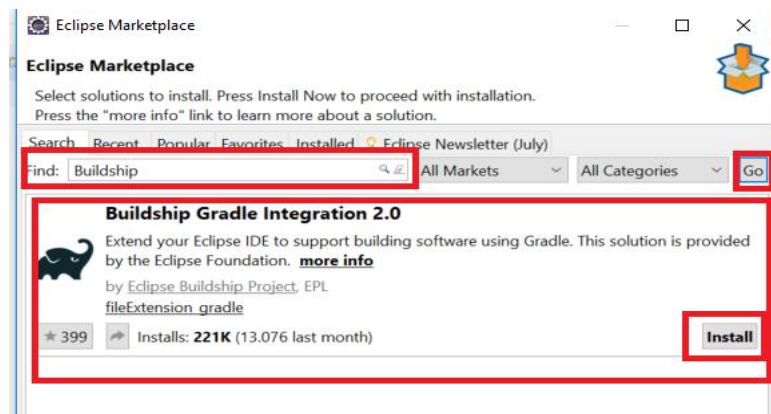


Figura 12 - Busca da ferramenta Buildship no Eclipse Marketplace

2.8. CRIANDO PROJETOS GRADLE

A ferramenta Eclipse fornece um assistente para a criação de projetos Gradle baseados em Java. Você pode alcançá-lo através do Arquivo ► Novo ► Outro conforme Figuras 10 e 11.

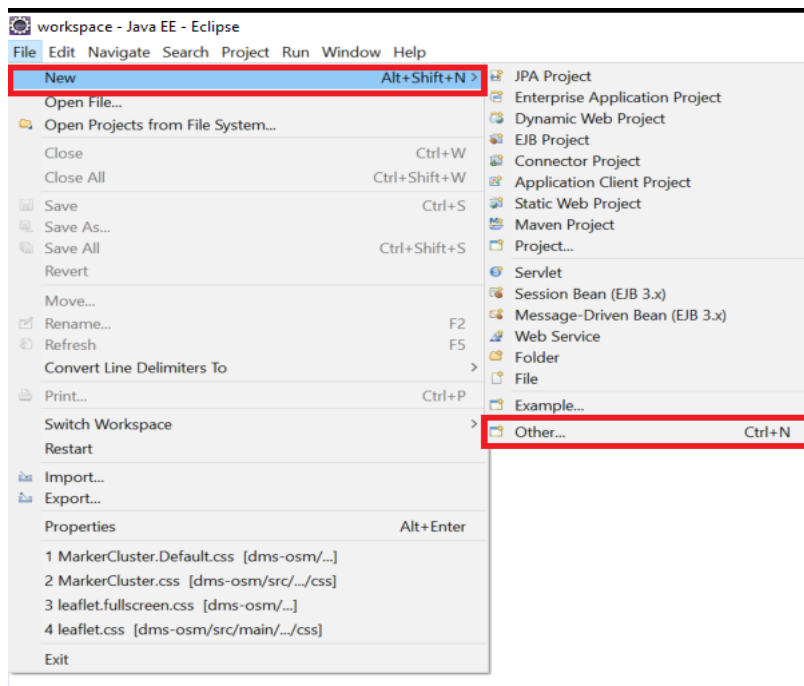


Figura 13 - Criação de um projeto Gradle

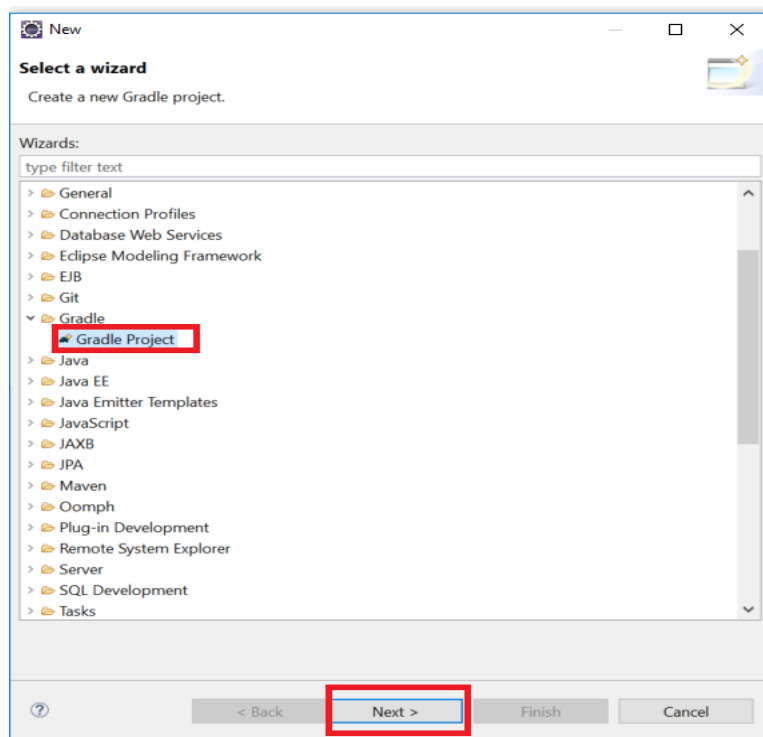


Figura 14 - Criação de um projeto Gradle

Preencha o *Project Name* com o nome desejado conforme Figura 12.

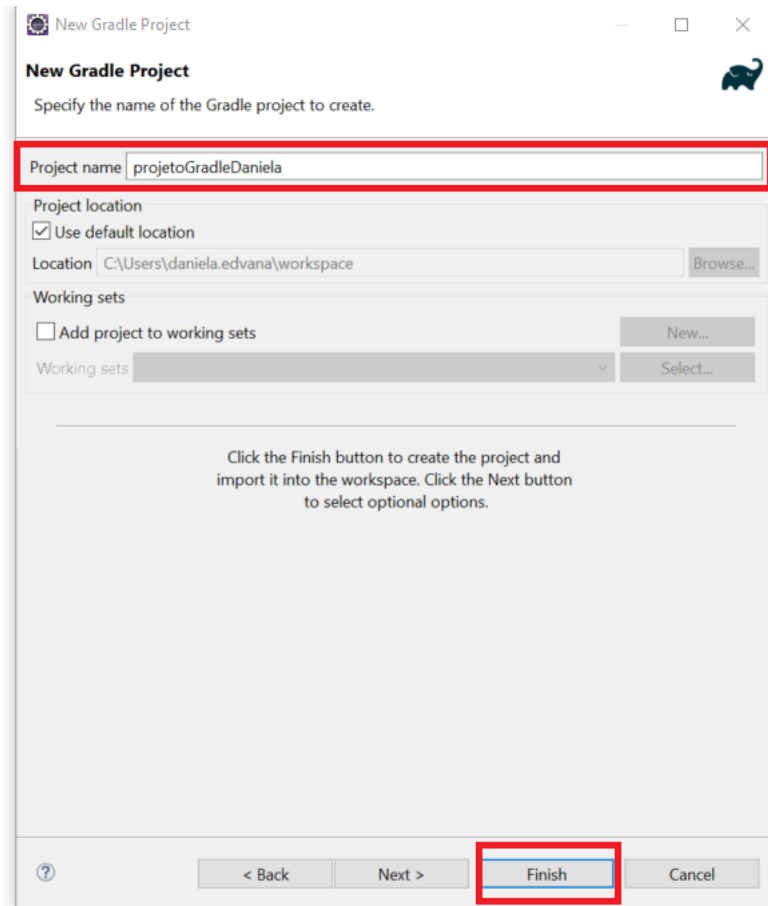


Figura 15 - Criação de um projeto Gradle

Pressione o botão Finalizar para criar o projeto. Isso desencadeia o comando `gradle init -ty java-library` e importa o projeto.

O projeto criado é semelhante à Figura 13.

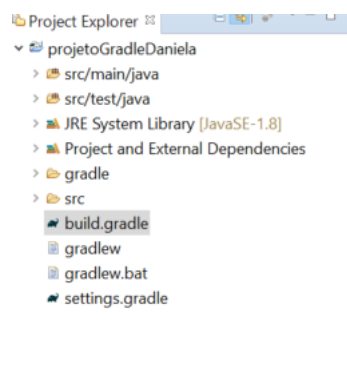


Figura 16 – Projeto Gradle

Prática 1 – Gerenciando dependências

Abra o MySQL Workbench e crie um banco de dados chamado dm107 utilizando a seguinte sql:

```
create database dm107;
```

Posteriormente execute a seguinte SQL:

```
use dm107;
```

Crie uma tabela chamada contato utilizando a seguinte SQL:

```
CREATE TABLE contato (  
    id int AUTO_INCREMENT PRIMARY KEY NOT NULL,  
    nome VARCHAR(300) NOT NULL,  
    email VARCHAR(300),  
    cel VARCHAR(50) NOT NULL  
);
```

Insira alguns dados na tabela contato utilizando as seguintes SQL:

```
insert into contato (nome, email, cel) values ("Aluno 1",  
"aluno1@inatel.br", "(35)8888-9999");  
insert into contato (nome, email, cel) values ("Aluno 2",  
"aluno2@inatel.br", "(35)7777-9999");  
insert into contato (nome, email, cel) values ("Aluno 3",  
"aluno3@inatel.br", "(35)6666-9999");  
insert into contato (nome, email, cel) values ("Aluno 4",  
"aluno4@inatel.br", "(35)2222-9999");
```

Abra o Eclipse e crie um projeto Gradle.

Substitua o conteúdo do arquivo build.gradle pelo seguinte código:

```
apply plugin: 'java'
```

```
// In this section you declare where to find the dependencies of  
your project
```

```
repositories {  
    mavenCentral()  
}
```

```
dependencies {  
    compile group: 'mysql', name: 'mysql-connector-java',  
version: '5.1.43'  
}
```

Para baixar a dependência clique com o botão direito sobre seu projeto e selecione a opção Gradle e posteriormente Refresh Gradle Project conforme Figura 14.

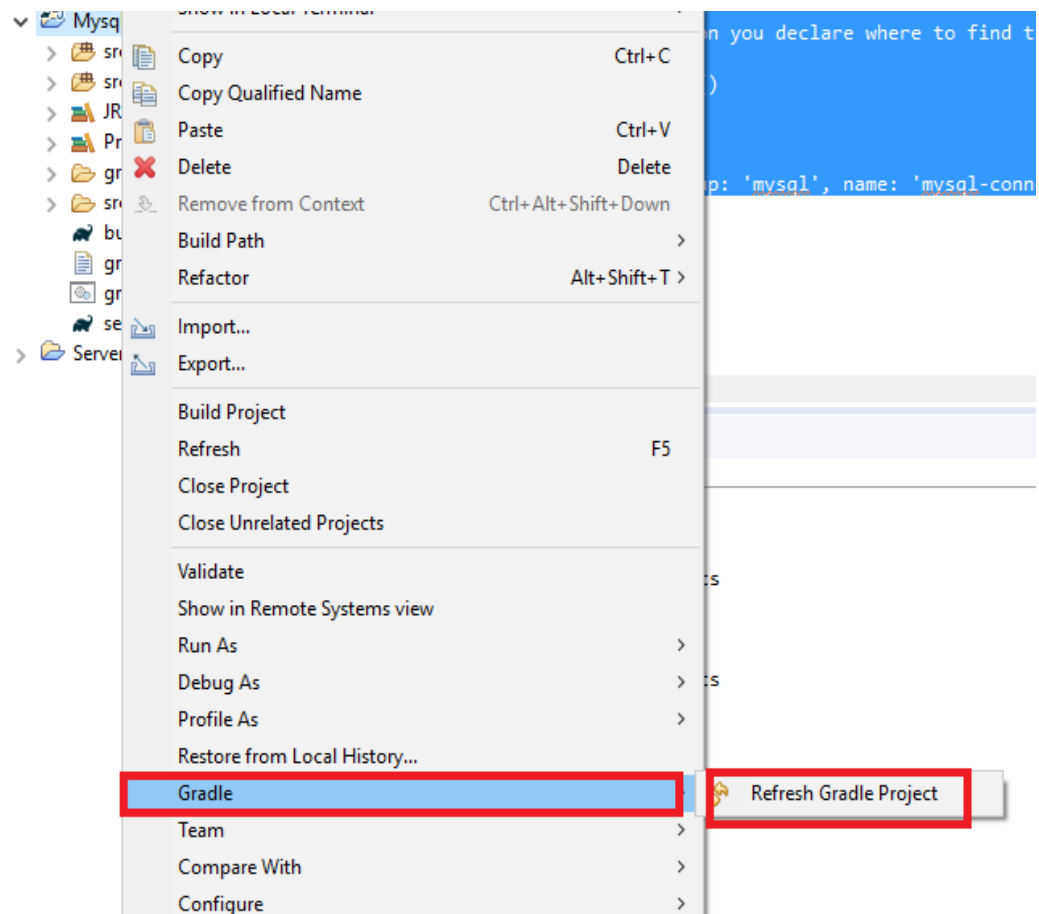


Figura 17 – Baixar as dependências em um projeto Gradle

Crie uma classe chamada ConnectionFactory contendo o seguinte código:

```
public class ConnectionFactory {  
    public Connection getConnection() {  
        try {  
            return DriverManager.getConnection(  
                "jdbc:mysql://localhost/dm107",  
                "root", "root");  
        } catch (SQLException e) {  
            throw new RuntimeException(e);  
        }  
    }  
}
```

Crie uma classe ConnectionTest contendo o seguinte código:

```

public static void main(String[] args) {
    ConnectionFactory connectionFactory = new
ConnectionFactory();
    Connection conn = connectionFactory.getConnection();
    ResultSet rs;
    try {
        rs = conn.prepareStatement("show
tables").executeQuery();
        while(rs.next()){
            String s = rs.getString(1);
            System.out.println(s);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

```

Para executar o projeto clique com botão direito e selecione a opção Run As e posteriormente Java Application conforme Figura 15.

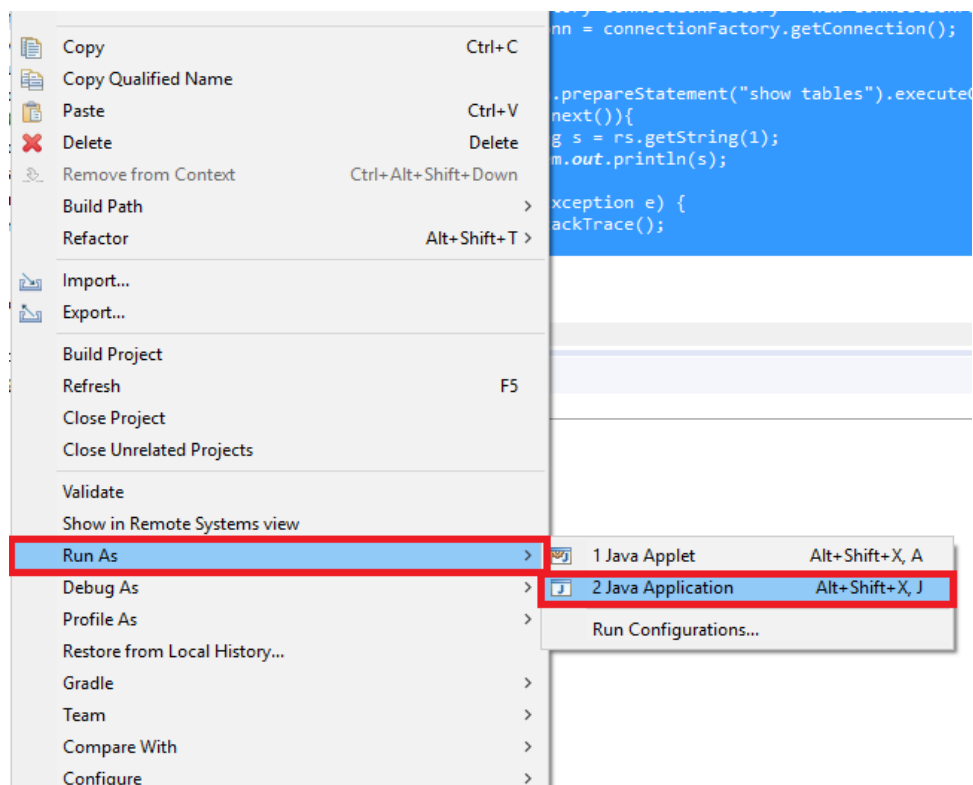


Figura 18 - Executar Projeto

Selecione a classe ConnectionTest conforme Figura 16 e clique no botão OK.

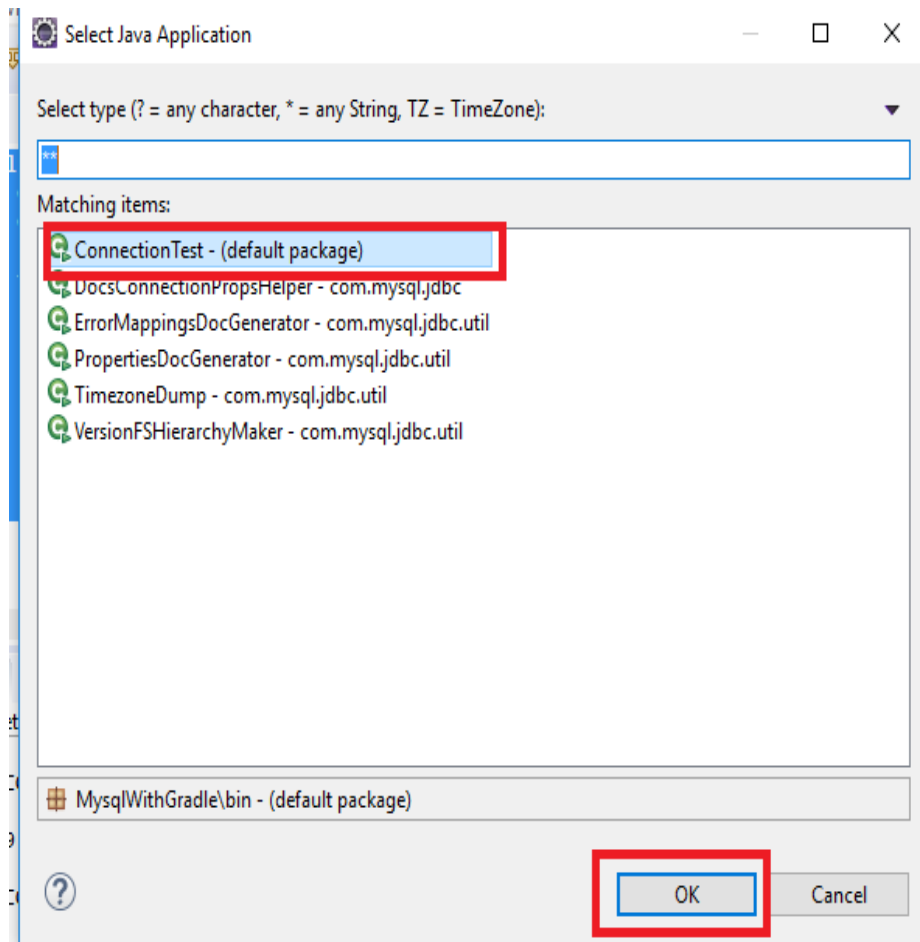


Figura 19 - Executar Projeto

O resultado esperado será apresentado no console como na Figura 17.

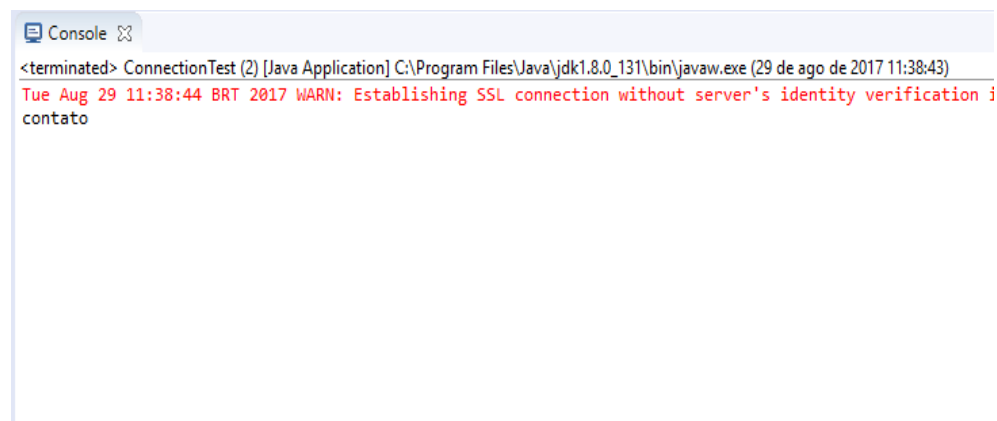


Figura 20 - Resultado da execução do projeto

Prática 2 – Criando subprojetos com Gradle

Passo 1 - Crie um projeto Gradle chamado GradleWithSubprojects.

Passo 2 - Remova as pastas e os arquivos destacados em vermelho conforme Figura 18.

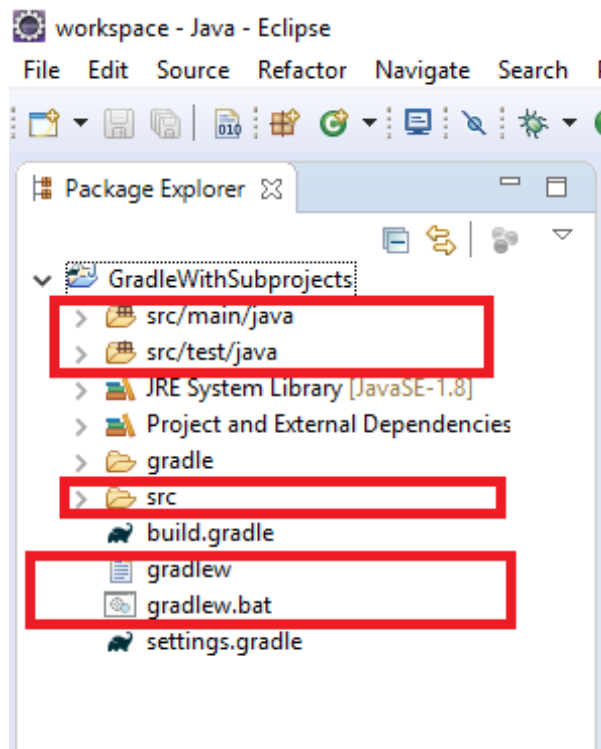


Figura 21 - Removendo arquivos na criação de um projeto Gradle

Passo 3 - Substitua o conteúdo do arquivo build.gradle pelo conteúdo abaixo:

```
subprojects {
    apply plugin : "java"
    repositories {
        mavenCentral()
    }
}
```

Passo 4 - Substitua o conteúdo do arquivo settings.gradle pelo conteúdo abaixo:

```
include 'UtilProject'
include 'ModelProject'
include 'PrincipalProject'
```

Passo 5 - Clique sobre o projeto com o botão direito, selecione a opção Gradle e posteriormente a opção Refresh Gradle Project

Os 3 Projetos definidos no arquivo settings.gradle serão criados automaticamente.

Passo 6 - Clique com botão direito sobre o projeto ModelProject selecione New > File.

Passo 7 - Preencha o campo filename com o seguinte valor build.gradle.

Passo 8 - No arquivo build.gradle criado coloque o conteúdo abaixo:

```
apply plugin: "java"
```

Passo 9 - Clique com botão direito sobre o projeto ModelProject selecione New > Source Folder.

Passo 10 - Preencha o campo Folder Name com o valor src/main/java.

Passo 11 - Clique sobre o source folder e crie um pacote chamado com.model.

Passo 12 - Crie uma classe ContatoModel e coloque o seguinte código:

```
public class ContatoModel {  
    private String name;  
    private String email;  
    private String telefone;  
  
    public ContatoModel() {}  
  
    public ContatoModel(String name, String email, String  
telefone) {  
        this.name = name;  
        this.email = email;  
        this.telefone = telefone;  
    }  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public String getEmail() {  
        return email;  
    }  
    public void setEmail(String email) {  
        this.email = email;  
    }  
    public String getTelefone() {  
        return telefone;  
    }  
    public void setTelefone(String telefone) {  
        this.telefone = telefone;  
    }  
}
```

Passo 13 - Clique com botão direito sobre o projeto UtilProject selecione New > File.

Passo 14 - Preencha o campo *Filename* com o seguinte valor build.gradle.

Passo 15 - No arquivo build.gradle criado coloque o conteúdo abaixo:

```
apply plugin: "java"
```

```

repositories {
    mavenLocal()
    mavenCentral()
}
dependencies {
    compile group: 'com.google.code.gson', name: 'gson',
version: '2.7'
}

```

Passo 16 - Clique com botão direito sobre o projeto ModelProject selecione New > Source Folder.

Passo 17 - Preencha o campo *Folder Name* com o valor src/main/java.

Passo 18 - Clique sobre o *source folder* e crie um pacote chamado com.util.

Passo 19 - Crie uma classe GsonUtil e coloque o seguinte código:

```

public abstract class GsonUtil {

    private static Gson gson = new
    GsonBuilder().setDateFormat("dd/MM/yyyy HH:mm:ss").create();

    public static <T> T fromToJson(String json, Class<T>
    classOfT) throws JsonSyntaxException {
        return gson.fromJson(json, classOfT);
    }

    public static String toJson(Object object) {
        return gson.toJson(object);
    }
}

```

Passo 20 - Clique sobre o projeto UtilProject com o botão direito, selecione a opção Gradle e posteriormente a opção Refresh Gradle Project.

A dependência da biblioteca Gson será baixada.

Passo 21 - Volte na classe GsonUtil e realize os *imports* necessários.

Passo 22 - Clique com botão direito sobre o projeto PrincipalProject selecione New > File.

Passo 23 - Preencha o campo *Filename* com o seguinte valor build.gradle.

Passo 24 - No arquivo build.gradle criado coloque o conteúdo abaixo:

```

apply plugin: 'java'

repositories {
    mavenLocal()
}

```

```

        mavenCentral()
    }
    dependencies {
        compile project(":ModelProject")
        compile project(":UtilProject")
    }
}

```

Passo 25 - Clique com botão direito sobre o projeto PrincipalProject selecione New > Source Folder.

Passo 26 - Preencha o campo *Folder Name* com o valor src/main/java.

Passo 27 - Clique sobre o *source folder* e crie um pacote chamado com.principal.

Passo 28 - Crie uma classe ConvertToJson e coloque o seguinte código:

```

public static void main(String[] args) {
    ContatoModel contatoModel = new ContatoModel("Aluno 1",
        "aluno1.inatel.br", "(35)9999-0000");

    String json = GsonUtil.toJson(contatoModel);

    System.out.println(json);
}

```

Passo 29 - Clique sobre o projeto PrincipalProject com o botão direito, selecione a opção Gradle e posteriormente a opção Refresh Gradle Project.

Passo 30 - Volte na classe ConvertToJson e realize os *imports* necessários.

Passo 31 - Clique sobre o a classe ConvertToJson com o botão direito, selecione a opção Run As e posteriormente a opção Java Application.

O resultado da execução do projeto será mostrado no console do eclipse conforme Figura 19.

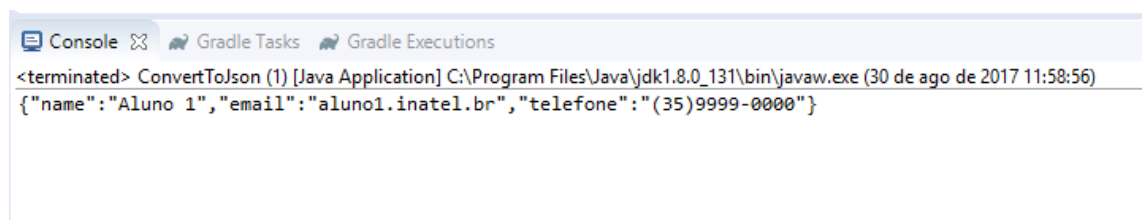


Figura 22 - Resultado da execução do projeto

Prática 3 – Criando projeto web com Gradle

Passo 1 - Crie um projeto Gradle chamado NovoAno.

Passo 32 - Remova as pastas e os arquivos destacados em vermelho conforme Figura 20.

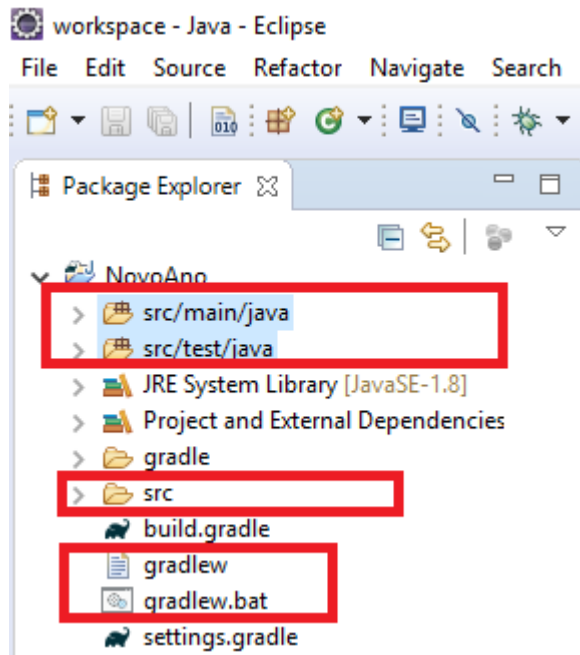


Figura 23- Removendo arquivos na criação de um projeto Gradle

Passo 3 - Substitua o conteúdo do arquivo build.gradle pelo conteúdo abaixo:

```
subprojects {
    apply plugin : "java"
    repositories {
        mavenCentral()
    }
}
```

Passo 4 - Substitua o conteúdo do arquivo settings.gradle pelo conteúdo abaixo:

```
include 'NovoAnoUtil'
include 'NovoAnoWeb'
```

Passo 5 - Clique sobre o projeto com o botão direito, selecione a opção Gradle e posteriormente a opção *Refresh* Gradle Project

Os 2 Projetos definidos no arquivo settings.gradle serão criados automaticamente.

Passo 6 - Clique com botão direito sobre o projeto NovoAnoUtil selecione New > File.

Passo 7 - Preencha o campo *filename* com o seguinte valor build.gradle.

Passo 8 - No arquivo build.gradle criado coloque o conteúdo abaixo:

```
apply plugin: "java"

dependencies {
    compile "joda-time:joda-time:2.9.4"
}
```

Passo 9 - Clique com botão direito sobre o projeto NovoAnoUtil selecione a opção Gradle e posteriormente a opção *Refresh* Gradle Project.

Passo 10 - Clique com botão direito sobre o projeto NovoAnoUtil selecione New > Source Folder.

Passo 11 - Preencha o campo Folder Name com o valor src/main/java.

Passo 12 - Clique sobre o *source folder* e crie um pacote chamado com.util.dm107

Passo 13 - Crie uma classe DateUtil e coloque o seguinte código:

```
public class DateUtil {  
    public int daysToNewYear() {  
        LocalDate fromDate = new LocalDate();  
        LocalDate newYear =  
fromDate.plusYears(1).withDayOfYear(1);  
        return Days.daysBetween(fromDate,  
newYear).getDays();  
    }  
}
```

Passo 14 - Volte na classe DateUtil e realize os *imports* necessários.

Passo 15 - Clique com botão direito sobre o projeto NovoAnoWeb selecione New > File.

Passo 16 - Preencha o campo *Filename* com o seguinte valor build.gradle.

Passo 17 - No arquivo build.gradle criado coloque o conteúdo abaixo:

```
apply plugin: 'java'  
apply plugin: "war"  
apply plugin: 'com.bmuschko.tomcat'  
  
repositories {  
    mavenLocal()  
    mavenCentral()  
}  
  
dependencies {  
    compile project(":NovoAnoUtil")  
    providedCompile "javax.servlet:javax.servlet-api:3.1.0"  
  
    def tomcatVersion = '7.0.57'  
    tomcat "org.apache.tomcat.embed:tomcat-embed-  
core:${tomcatVersion}",  
    "org.apache.tomcat.embed:tomcat-embed-logging-  
juli:${tomcatVersion}"
```

```

        tomcat("org.apache.tomcat.embed:tomcat-embed-
jasper:${tomcatVersion}") {
            exclude group: 'org.eclipse.jdt.core.compiler',
module: 'ecj'
        }
    }
    buildscript {
        repositories {
            mavenCentral()
        }
        dependencies {
            classpath 'com.bmuschko:gradle-tomcat-plugin:2.0'
        }
    }
}

```

Passo 18 - Clique com botão direito sobre o projeto NovoAnoWeb selecione New > Source Folder.

Passo 19 - Preencha o campo *Folder Name* com o valor src/main/java.

Passo 20 - Clique sobre o *source folder* e crie um pacote chamado com.web.dm107.

Passo 21 - Crie uma classe DaysToGoServlet e coloque o seguinte código:

```

public class DaysToGoServlet extends HttpServlet {

    public void doGet(HttpServletRequest req,
        HttpServletResponse resp)
        throws IOException {
        resp.setContentType("text/plain");
        resp.getWriter().println(new
        DateUtil().daysToNewYear());
    }
}

```

Passo 22 - Clique sobre o projeto NovoAnoWeb com o botão direito, selecione a opção Gradle e posteriormente a opção Refresh Gradle Project.

Passo 23 - Volte na classe NovoAnoWeb e realize os *imports* necessários.

Passo 24 – Clique sobre o *folder source/main* e crie um novo *folder* chamado webapp conforme Figuras 21 e 22.

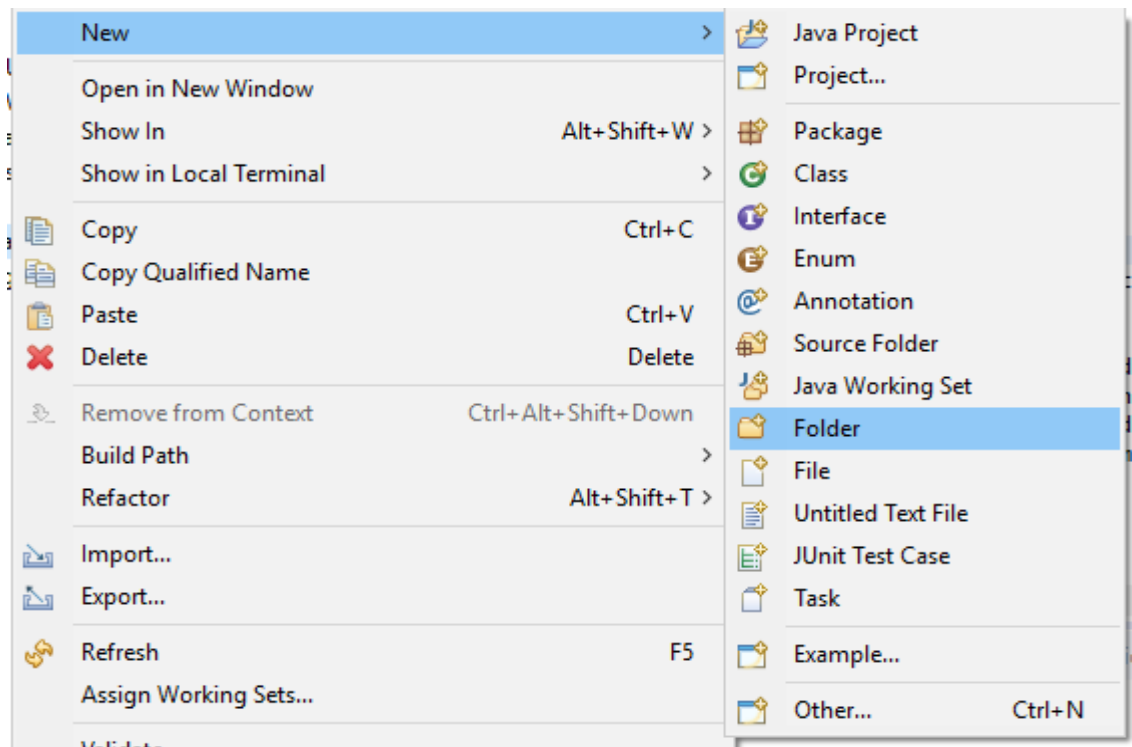


Figura 24 - Criando o *folder* webapp

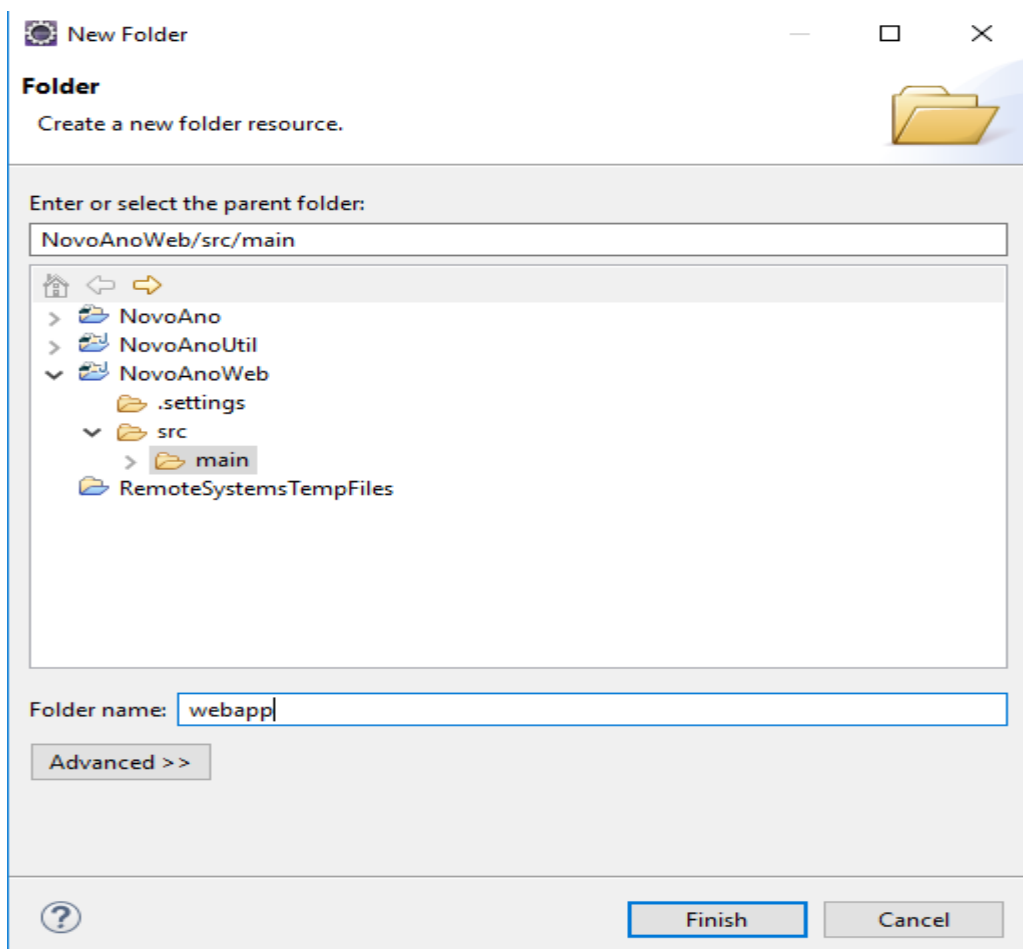


Figura 25 - Criando o *folder* webapp

Passo 25 – Clique sobre o *folder webapp* e crie um novo *folder* chamado WEB-INF.

Passo 26 – Clique sobre o *folder* WEB-INF e crie um arquivo chamado web.xml e coloque o seguinte código:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
version="2.5">
    <servlet>
        <servlet-name>DaysToGo</servlet-name>
        <servlet-
class>com.web.dm107.DaysToGoServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>DaysToGo</servlet-name>
        <url-pattern>/daystogo</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

Passo 27 – Clique sobre o *folder webapp* e crie um arquivo chamado index.jsp e coloque o seguinte código:

```
<jsp:useBean id="days" class="com.util.dm107.DateUtil"/>
<html>
    <body>
        <p><%=days.daysToNewYear() %> days to go this year.</p>
    </body>
</html>
```

Para executar o projeto web com Tomcat execute os seguintes passos:

Passo 28 – Abra a aba Gradle Tasks conforme Figura 23 e 24:

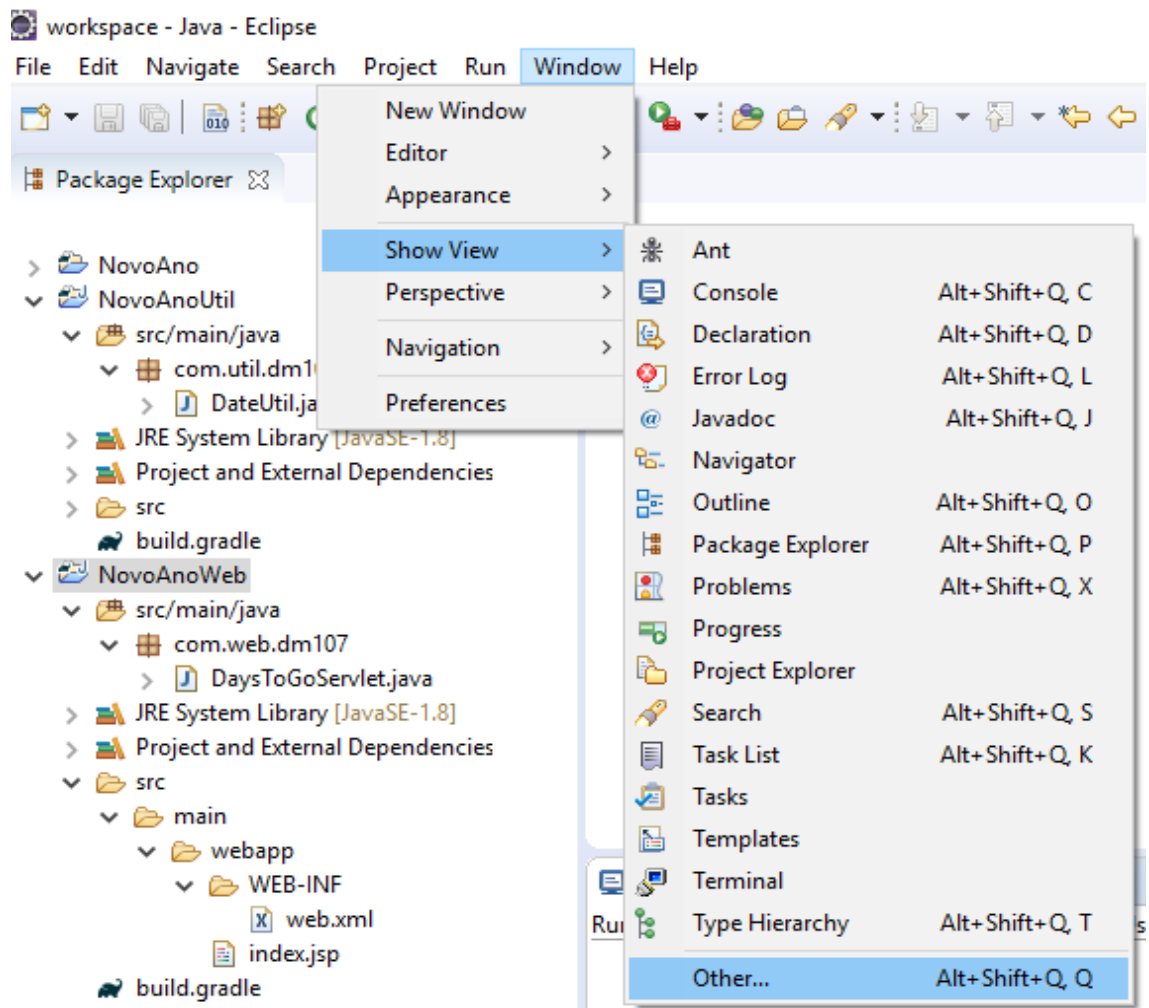


Figura 26 - Abrindo a aba Gradle Tasks

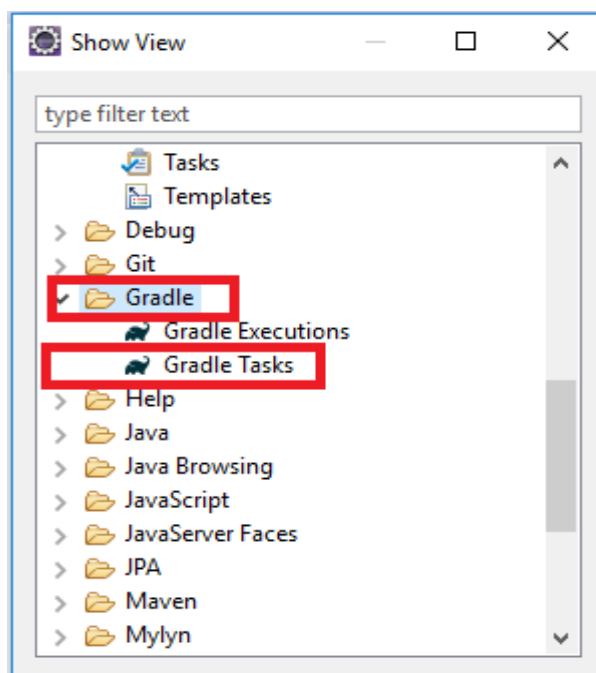


Figura 27 - Abrindo a aba Gradle Tasks

Passo 29 – Na aba Gradle Tasks clique com botão direito sobre a opção *build* e selecione a opção *Run Gradle Tasks* conforme Figuras 25 e 26:

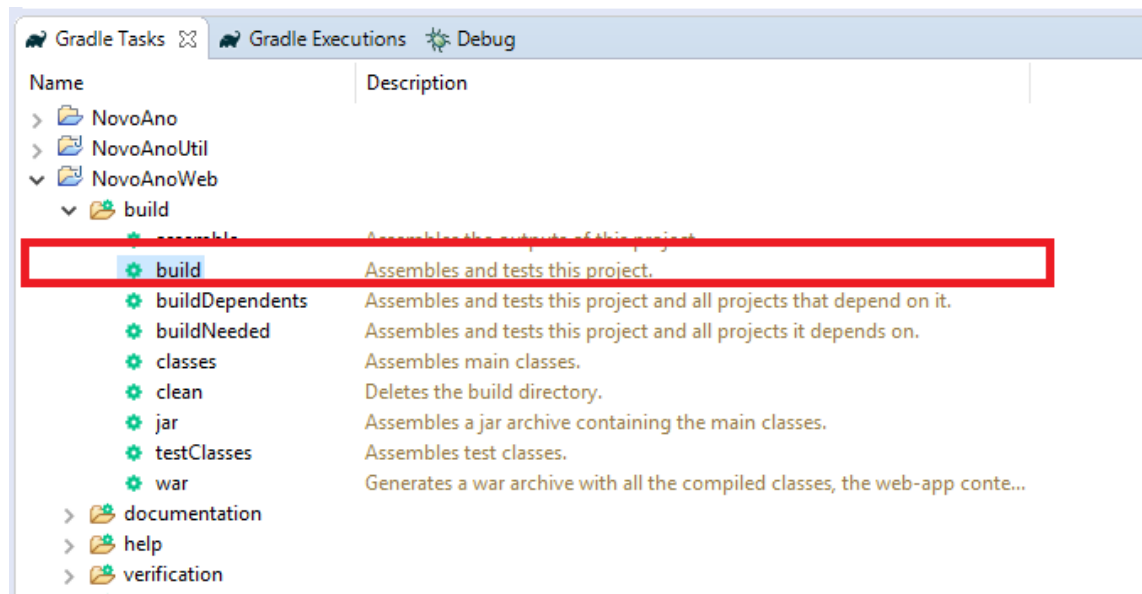


Figura 28- Run Gradle Tasks

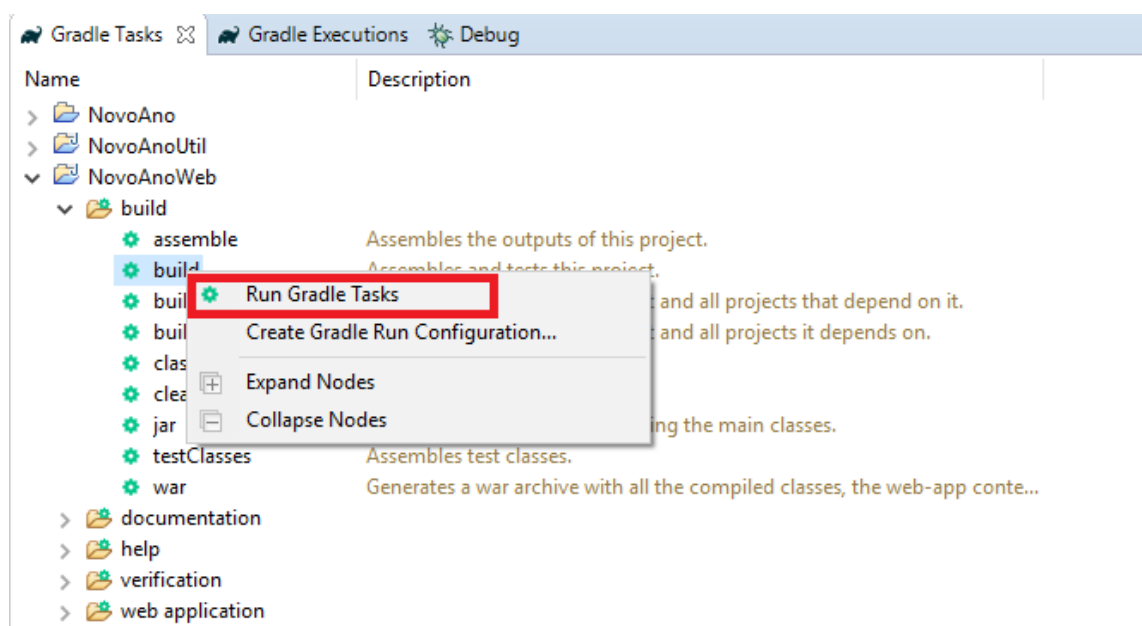


Figura 29 - Run Gradle Tasks

Passo 30 – Verifique a aba Gradle Executions, você deve ver uma lista de tarefas executadas conforme Figura 27.

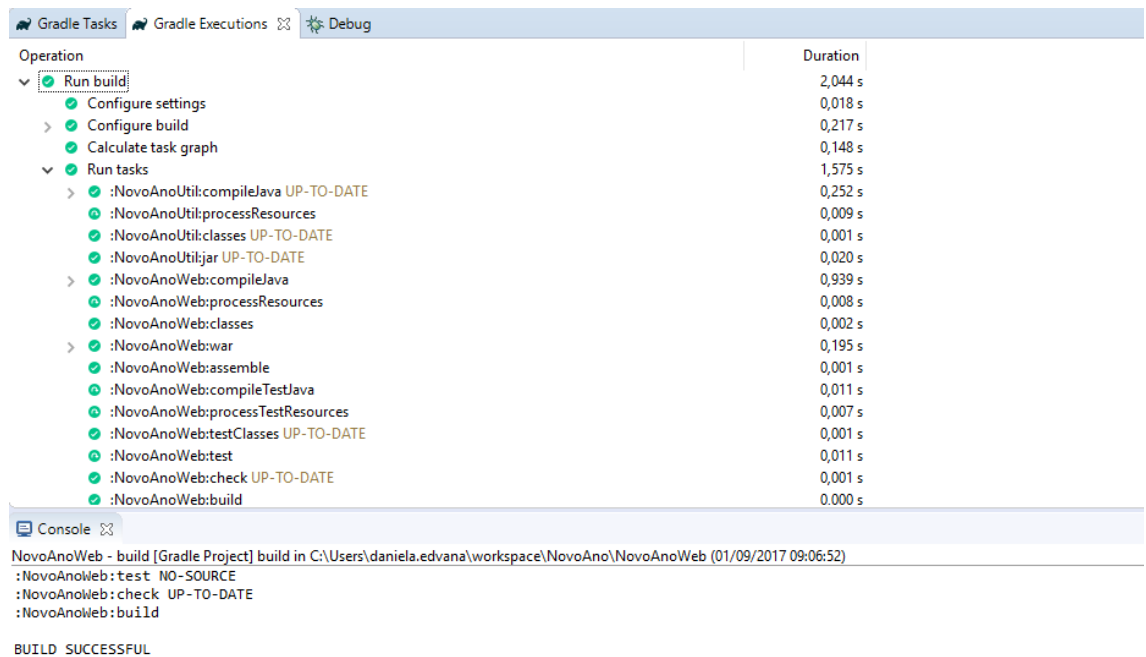


Figura 30- Execução do Run Gradle Tasks

Passo 31 - Clique sobre o projeto NovoAnoWeb com o botão direito, selecione a opção Run e posteriormente a opção Run Configurations.

Passo 32 – Clique com botão direito sobre a opção Gradle Project e Selecione New conforme Figura 28.

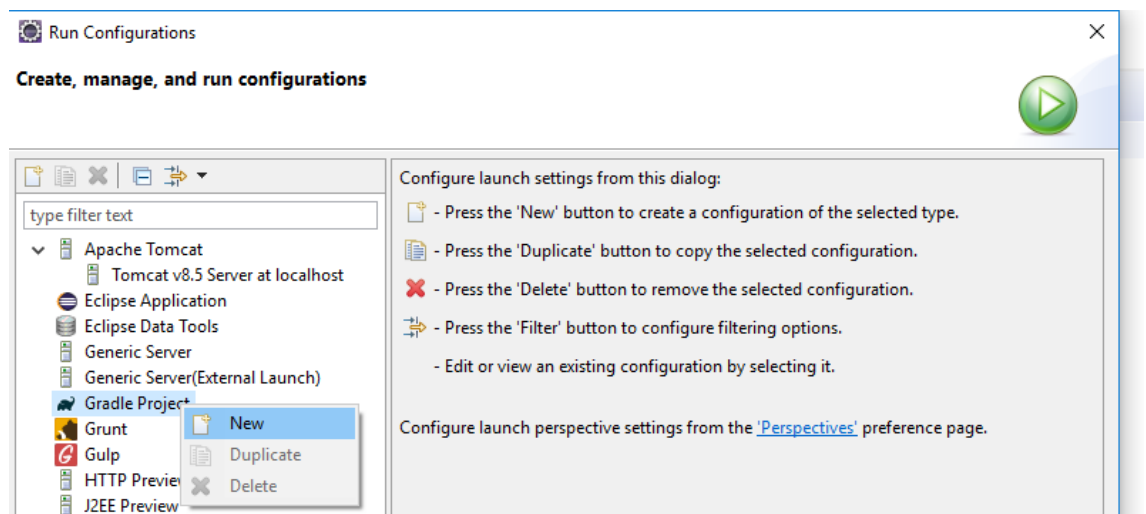


Figura 31 - Configuração para execução de um projeto Gradle Web

Passo 33 – Preencha os campos conforme Figura 29.

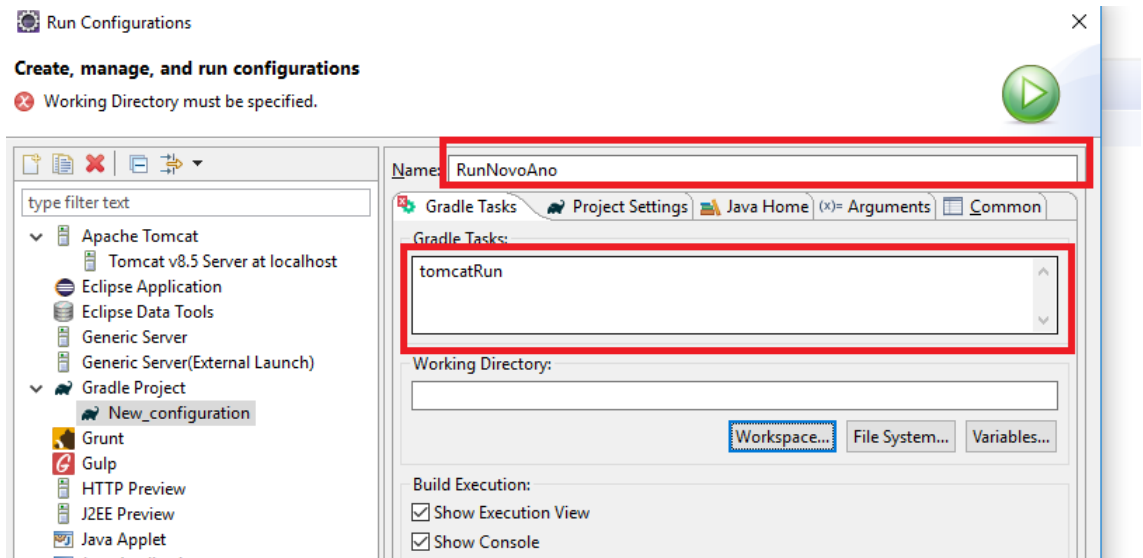


Figura 32 - Configuração para execução de um projeto Gradle Web

Passo 34 – Clique em *workspace* e selecione onde seu projeto web está salvo. O campo *Working Directory* será preenchido conforme Figura 30.

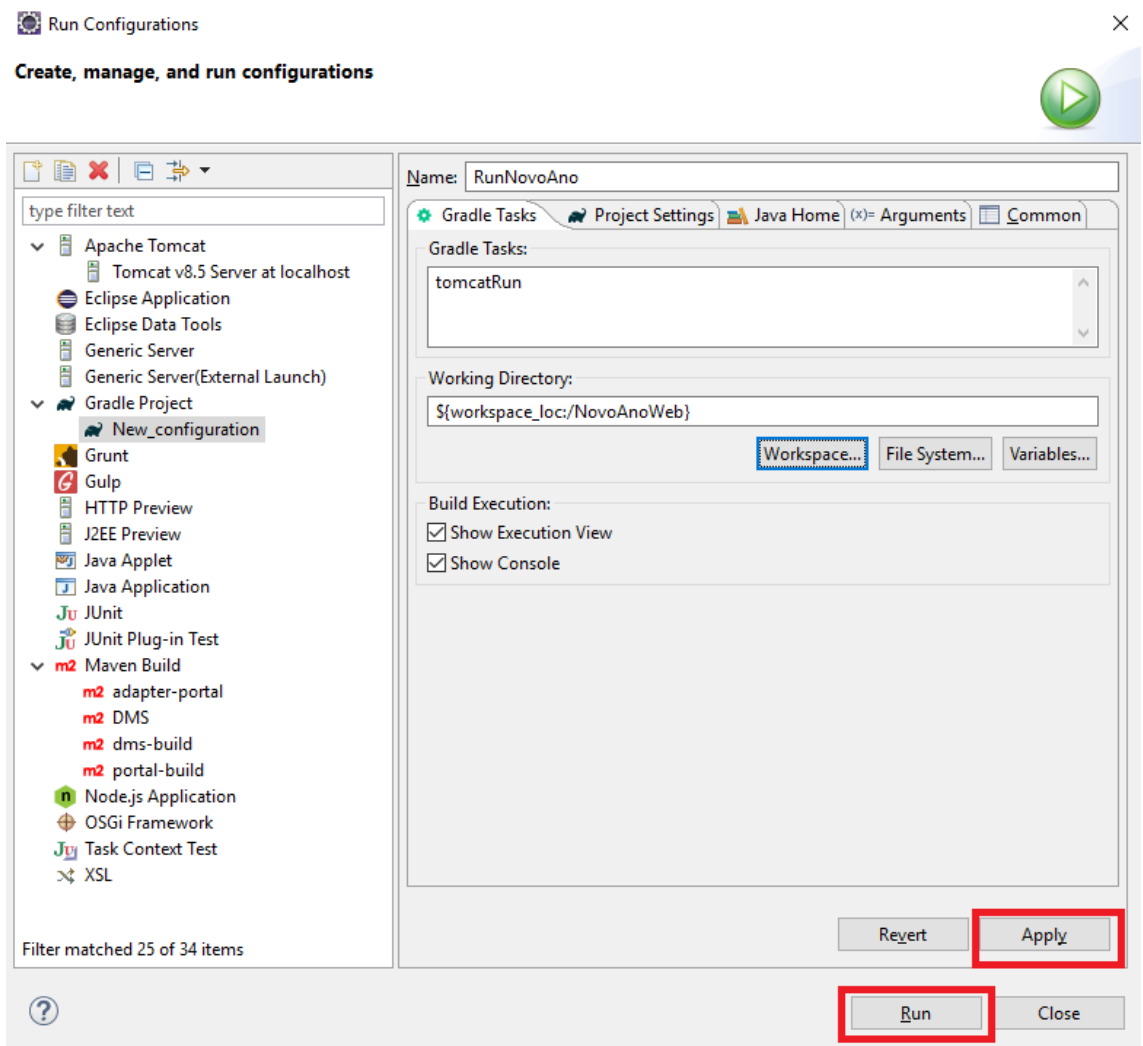


Figura 33 - Configuração para execução de um projeto Gradle Web

Passo 35 - O resultado será mostrado no console do eclipse conforme Figura 31.

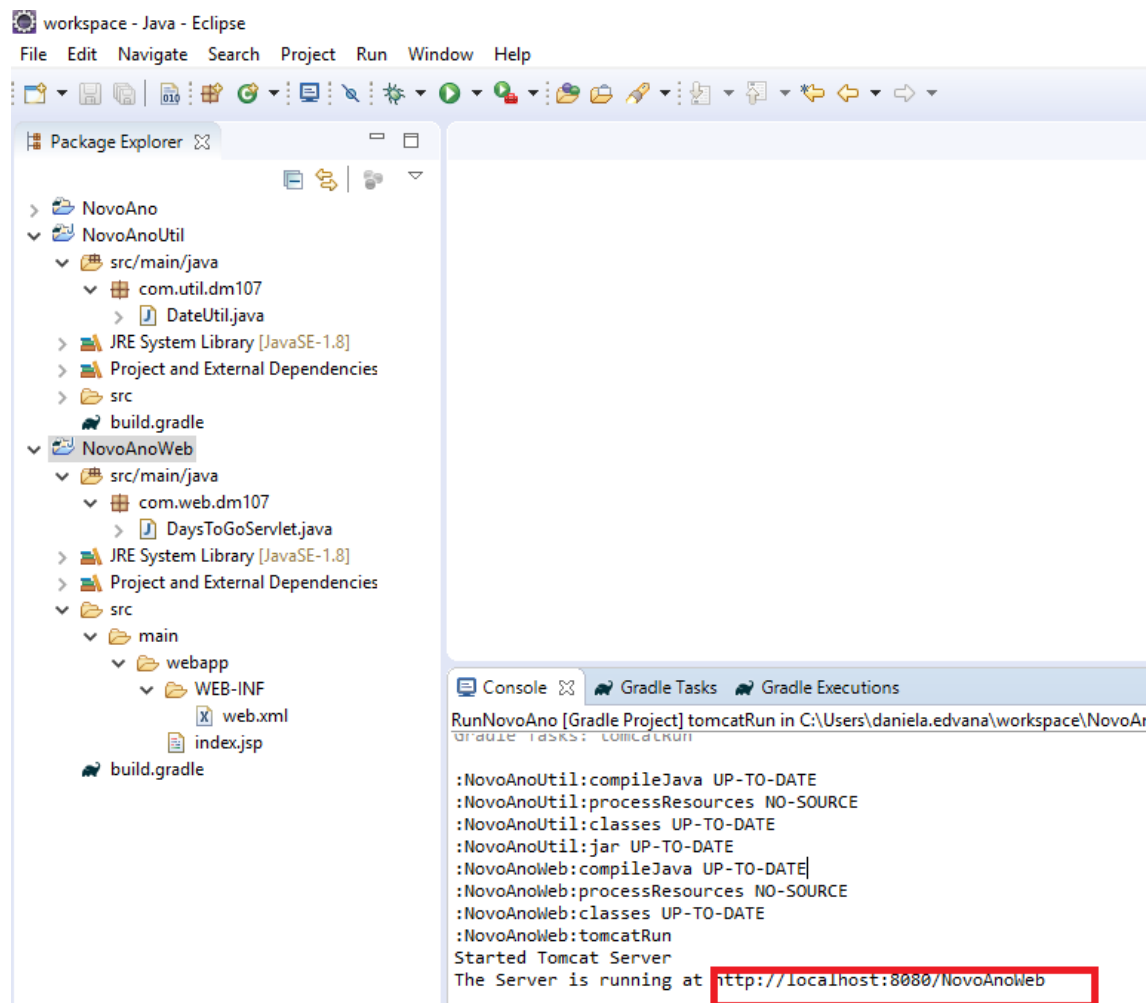


Figura 34 - Execução do projeto NovoAnoWeb

32. Passo 35 – Acesse o *URL* mostrada no console em seu *browser* conforme Figura



Figura 35 - Acessando o projeto NovoAnoWeb no browser

REFERÊNCIAS

Create a Gradle Java Web Application and run on Gradle Tomcat Plugin.

Disponível em <<http://o7planning.org/en/11247/create-a-gradle-java-web-application-and-run-on-gradle-tomcat-plugin>> Acesso em 15 Ago. 2017.

Kurose, James F.; Ross, Keith W. **Redes de computadores e a internet:** Uma abordagem top-down. 5. ed. São Paulo: Addison Wesley, 2010. Tradução de: Computer networking a top-down approach feauting the Internet, 5th ed

Gradle Build Tool. Disponível em <<https://gradle.org/>>. Acesso em 15 Ago. 2017.

Muschko, Benjamin. **Gradle in Action.** Manning Publications, 2014.

Ruby, Sam.; Richardson, Leonard. **Restful web services.**O'Reilly, 2007.

Saudate. Alexandre. **Rest Construa API's Inteligentes de Maneira Simples.** Casa do Código.

Scholz, Simon.; Lars, Vogel. **Using the Gradle build system in the Eclipse IDE – Tutorial.** Disponível em <

<http://www.vogella.com/tutorials/EclipseGradle/article.html#eclipse-gradle-support>>

Acesso em 15 Ago. 2017.

Souza, Jaime Freire. **Avaliação do Protocolo HTTP 2.0.** Disponível em <

<http://www2.uesb.br/computacao/wp-content/uploads/2014/09/MONOGRAFIA-JAIME-FINAL.pdf>>. Acesso em 15 Ago. 2017.