Gradle (Acelere la productividad del desarrollador)

Objetivo

Conocer y entender el uso de Gradle como herramienta a la hora de construir software.

Población

Dirigido a personas de cualquier edad y de cualquier nivel académico con conocimientos en desarrollo de software.

Actividades

El curso tiene una duración de 8 horas y puede ser desarrollado en cualquier lugar que cuente con un espacio cómodo para los asistentes, posea algún instrumento para proyectar como un televisor o un video bean. Cada asistente debe contar con un portátil, el cual debe tener instalado la máquina virtual de java, el IDE de su elección el cual debe contar con soporte para Gradle (Se recomienda utilizar Eclipse o IntelliJ IDEA).

Actividad	**Descripción**
Que es?	En este espacio el asistente comprenderá realmente que es Gradle
Características	El asistente conocerá una a una las característica de Gradle con el fin de conocer que se puede y no hacer con él
Descargar e Instalar	El asistente aprenderá a descargar la última versión o versiones anteriores de Gradle según su elección y a instalarla
Obtener ayuda y licencia	Se le indicará al asistente los sitios oficiales de ayuda y documentación acerca de Gradle y los por menores de la licencia que utiliza Gradle en su distribución
Gradle vs Maven	El asistente aprenderá las ventajas y diferencias que tiene Gradle sobre Maven
Migrando desde Maven	El asistente aprenderá a migrar un proyecto de Maven a Gradle
Interfaz de línea de comandos	El asistente reconocerá y aprenderá los principales comando que puede utilizar con Gradle en su línea de comandos
Gradle Wrapper	El asistente aprenderá porque Gradle Wrapper es la forma recomendada de ejecutar cualquier construcción de Gradle.
Gradle con Java	El asistente llevará acabo una práctica donde se utilizara Gradle con el lenguaje de programación Java para construir una aplicación.

Desarrollo

1. **Que es?**

Gradle es una herramienta de construcción. Segun la documentación oficial Gradle es una herramienta de automatización de compilación de código abierto centrada en la **flexibilidad** y el **rendimiento**. Los scripts de compilación de Gradle se escriben usando Groovy o Kotlin DSL(Lenguaje de Dominio Específico). Algunas de sus características más importantes son:

- **Altamente personalizable**
- **Rápido**
- **Potente**

2. **Características**

- Ejecución de Gradle Builds
    - Rendimiento
        - **Construcciones incrementales**: Gradle comprueba entre ejecuciones de compilación si la entrada, salida o implementación de una tarea ha cambiado desde la última invocación de compilación. De lo contrario, la tarea se considera actualizada y no se ejecuta.
        - **Caché de resultados de tareas**
        - **Subtareas incrementales**: Cuando Gradle descubre que la entrada o salida de una tarea ha cambiado entre ejecuciones de compilación, la tarea se ejecuta nuevamente. La tarea puede usar la API incremental para saber qué archivos han cambiado exactamente. Con esta información, la tarea puede no necesitar reconstruir todo.
        - **Compiler Daemon**
        - **Ejecución paralela**
        - **Descarga paralela de dependencias**
    - Build Scans
    - Interfaz de línea de comando
        - **Exclusión de tarea**
        - **Construcción continua**
        - **Compilaciones compuestas**
        - **Dry Run**:Ejecute una compilación para ver qué tareas realmente se ejecutan sin ejecutar las acciones de la tarea.
        - **Continuar la ejecución después de las fallas**
        - **Sincronización caché de dependencias con repositorio**
- Creación de compilaciones de Gradle

- Gestión de Dependencia
  - **Dependencias transitivas**
  - **Áreas de dependencia personalizadas**
  - **Diseño de repositorio personalizado**
  - **Dependencias basadas en archivos**
  - **Caché de Dependencia de terceros**
  - **Lee el formato de metadatos POM**
  - **Lee el formato de metadatos Ivy**
  - **Dependencias dinámicas**
  - **Bloqueo de Dependencia Dinámica**
  - **Reglas de selección de dependencias dinámicas**
  - **Versión de resolución de conflictos**
  - **Versión de resolución de conflictos**
  - **Soporte mejorado de resolución de metadatos**
  - **Reemplazo de dependencias externas y de proyecto**
- Estandarización de Gradle en equipos
  - **Entorno de compilación de autoaprovisionamiento**
  - **Configuración del entorno de compilación controlado por versión**
  - **Distribuciones personalizadas**: Cada distribución de Gradle tiene un directorio init.d en el que puede colocar scripts personalizados que pre configuran su entorno de compilación.
- Modelado de dominio de software
  - **Contenedores de objetos de dominio**
  - **Orden de tareas avanzadas**
  - **Inferencia de Dependencia de Tarea**
  - **Finalizadores de tareas**
  - **Creación dinámica de tareas**
  - **Escuchador de eventos de compilación de granularidad fina**
  - **Inyección de comportamiento basado en el usuario**
  - **Inyección de comportamiento por compilación**

Gradle contiene muchas otras funciones especialmente para JVM, Android, C ++, Swift, Objective C y otros ecosistemas.

3. Descargar e Instalar

Prerrequisitos: Gradle se ejecuta en todos los sistemas operativos principales y solo requiere una Java JDK versión 7 o superior para ejecutarse. Gradle se envía con su propia biblioteca Groovy, por lo tanto, Groovy no necesita ser instalado. Gradle ignora cualquier instalación existente de Groovy. Gradle usa cualquier JDK que encuentre en su camino.

Alternativamente, puede configurar la variable de entorno **JAVA_HOME** para que apunte al directorio de instalación del JDK deseado.

Descarga e instalación: Puede instalar Gradle en Linux, macOS o Windows. Gradle puede instalarse utilizando un administrador de paquetes (en este caso dichos administradores descargan e instalan Gradle):

- **SDKMAN**: Es una herramienta para administrar versiones paralelas de múltiples kits de desarrollo de software en la mayoría de los sistemas basados en Unix.
  - ❯ sdk install gradle
- **Homebrew**: es "el administrador de paquetes faltante para macOS".
  - ❯ brew install gradle
- **Scoop**: es un instalador de línea de comandos para Windows inspirado en Homebrew.
  - ❯ scoop install gradle
- **Chocolatey**: es "el administrador de paquetes para Windows".
  - ❯ choco install gradle
- **MacPorts**: es un sistema para administrar herramientas en macOS.
  - ❯ sudo port install gradle

Así como la instalación manual en ese caso debemos realizar varios pasos, los pasos a continuación seran realizado para el sistema operativo Windows teniendo en cuenta que es el sistema operativo utilizado por la mayoria de los asistentes, para los otros sistemas operativos se debe revisar la siguiente guia https://docs.gradle.org/current/userguide/installation.html:

- **Primer Paso:** Descargar Gradle, en cuyo caso se descargara un archivo de distribución .ZIP que viene en dos formas **binary-only** o **complete** (con documentos y fuentes).

  Para descargar la última distribución de Gradle ingresamos a https://gradle.org/releases/?_ga=2.129861772.1628691413.1532873854-1928928156.1519739528 donde encontraremos un listado con las últimas versiones en donde la primer es la versión más reciente en este momento es la 4.9.

Como se observa en la imagen anterior ses encuentra el titulo **Download:** seguido del link de descarga de las dos formas de distribución, escoge el que necesites. ¿Necesitas trabajar con una versión anterior? en ese caso solo sigue bajando el listado de versiones y escoge la versión que necesites.

- **Segundo Paso**: Se Crea un nuevo directorio en C:\Gradle, abra una ventana del Explorador de archivos y vaya al directorio donde se descargó la distribución de Gradle. Haga doble clic en el archivo ZIP para exponer el contenido. Arrastre la carpeta de contenido gradle-4.9 a su carpeta C:\Gradle recién creada. Alternativamente, puede descomprimir el ZIP de distribución de Gradle en C:\Gradle utilizando una herramienta de archivado de su elección.

- **Tercer Paso**: Para ejecutar Gradle, primero agregue la variable de entorno **GRADLE_HOME**. Esto debería apuntar a los archivos desempacados. A continuación, agregue **GRADLE_HOME/bin** a su variable de entorno **PATH**. Por lo general, esto es suficiente para ejecutar Gradle.

  Cómo agregar una variable de entorno?. En el Explorador de archivos , haga clic derecho en el ícono **This PC**, luego haga clic en Properties → Advanced System Settings→ Environmental Variables.

  En System Variables puedes agregar una nueva variable de entorno, también podrías seleccionar **Path**, haz clic **Edit**. agrega una entrada para C:\Gradle\gradle-4.9\bin.

  Haga clic en Aceptar para guardar.

NOTA: El uso de Gradle Wrapper es la forma recomendada de actualizar Gradle.

4. Obtener ayuda y licencia

Foro: Gradle cuenta con un foro en el cual puedes hacer preguntas o contestarlas https://discuss.gradle.org/?_ga=2.193765421.1628691413.1532873854-1928928156.1519739528

Capacitación: Gradle proporciona capacitación gratuita desde su página de entrenamiento https://gradle.org/training/?_ga=2.234209694.1628691413.1532873854-1928928156.1519739528, solo debes registrarte.
Licencias: El código fuente de Gradle está abierto y tiene licencia **Apache License 2.0**. El manual de usuario de Gradle y las referencias de DSL están autorizadas bajo la licencia internacional **Creative Commons Attribution-NonCommercial-ShareAlike 4.0**.

5. Gradle vs Maven

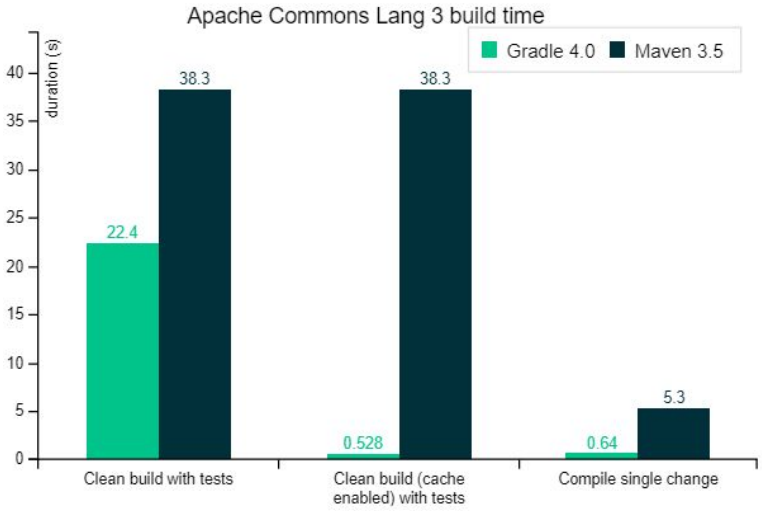
las principales diferencias entre Gradle y Apache Maven son:

- **Flexibilidad:** Gradle se modela de una manera que es extensible de las formas más fundamentales. El modelo de Gradle también permite su uso para el desarrollo nativo con C / C ++ y se puede ampliar para abarcar cualquier ecosistema. Tanto Gradle como Maven proporcionan convención sobre la configuración. Sin embargo, Maven proporciona un modelo muy rígido que hace que la personalización sea tediosa y, a veces, imposible. Si bien esto puede facilitar la comprensión de cualquier compilación de Maven, siempre que no tenga requisitos especiales, también lo hace inadecuado para muchos problemas de automatización. Gradle, por otro lado, está diseñado pensando en un usuario responsable.

- **Rendimiento:** Tanto Gradle como Maven emplean algún tipo de construcción de proyecto paralelo y resolución de dependencia paralela. Las mayores diferencias son los mecanismos de Gradle para evitar el trabajo e incrementar la productividad. Las 3 características principales que hacen que Gradle sea mucho más rápido que Maven son:

  Incrementalidad: Gradle evita el trabajo al rastrear la entrada y salida de tareas y solo ejecuta lo que es necesario, y sólo procesa los archivos que cambiaron cuando fue posible.
  Build Cache: Reutiliza las salidas de construcción de cualquier otra construcción de Gradle con las mismas entradas, incluso entre máquinas.

Gradle Daemon: Un proceso de larga duración que mantiene la información de compilación "caliente" en la memoria.



Estas y otras características de rendimiento hacen que Gradle sea al menos el doble de rápido en casi todos los escenarios (100 veces más rápido para grandes construcciones que usan la memoria caché de compilación) en esta comparación de rendimiento Gradle vs Maven .

- **Experiencia del usuario:** La tenencia más larga de Maven significa que su soporte a través de IDEs es mejor para muchos usuarios. El soporte IDE de Gradle continúa mejorando rápidamente, sin embargo. Por ejemplo, Gradle ahora tiene una DSL basada en Kotlin que proporciona una experiencia IDE mucho mejor. El equipo de Gradle está trabajando con los fabricantes de IDE para mejorar mucho la compatibilidad de edición.

  Aunque los IDEs son importantes, una gran cantidad de usuarios prefieren ejecutar operaciones de compilación a través de una interfaz de línea de comandos. Gradle proporciona una CLI moderna que tiene características de descubrimiento como **gradle tasks**, así como un mejor registro y finalización de línea de comandos.

  Finalmente, Gradle proporciona una interfaz de usuario interactiva basada en web para depurar y optimizar construcciones: escaneos de compilación.

- **Gestión de dependencias:** Ambos sistemas de compilación ofrecen capacidad integrada para resolver dependencias de repositorios configurables. Ambos pueden almacenar en caché las dependencias localmente y descargarlas en paralelo. Como consumidor de la

biblioteca, Maven permite anular una dependencia, pero solo por versión. Gradle proporciona selección de dependencias personalizables y reglas de sustitución que se pueden declarar una vez y manejar dependencias no deseadas en todo el proyecto. Este mecanismo de sustitución permite a Gradle construir múltiples proyectos fuente juntos para crear compilaciones compuestas.

Maven tiene pocos ámbitos de dependencia incorporados, lo que obliga a las arquitecturas de módulos poco prácticas en escenarios comunes, como el uso de dispositivos de prueba o generación de código. No hay separación entre pruebas unitarias y las pruebas de integración, por ejemplo. Gradle permite ámbitos de dependencia personalizados, lo que proporciona versiones mejor modeladas y más rápidas.

Como productor de biblioteca, Gradle permite a los productores declarar las dependencias api o implementation para evitar que las bibliotecas no deseadas se filtren en los classpaths de los consumidores. Maven permite a los editores proporcionar metadatos a través de dependencias opcionales .

6. Migrando desde Maven

Conversión automatizada: Gradle y Maven tienen puntos de vista fundamentalmente diferentes sobre cómo construir un proyecto. Gradle se basa en un gráfico de dependencias de tareas , donde las tareas hacen el trabajo. Maven usa un modelo de fases fijas y lineales a las que puedes agregar objetivos (las cosas que hacen el trabajo). A pesar de esto, las migraciones pueden ser sorprendentemente fáciles porque Gradle sigue muchas de las mismas convenciones que Maven y la administración de dependencias funciona de manera similar.

Conversión automatizada: La tarea de Gradle init no solo le permitirá crear un nuevo esqueleto del proyecto, sino que también convertirá automáticamente un Maven existente en Gradle. Todo lo que tienes que hacer es ejecutar el comando:

> gradle init

Desde el directorio raíz del proyecto y dejar que Gradle haga su trabajo. Básicamente consiste en analizar los POM existentes y generar los correspondientes archivos de compilación de Gradle más un archivo settings.gradle si se trata de una compilación de varios proyectos.

Bills of Materials (BOMs): Maven permite compartir restricciones en dependencias mediante la definición de dependencias dentro de la sección

de gestión de dependencias en un archivo POM con **<packaging>pom</packaging>**. Este tipo especial de POM (una lista de materiales) se puede importar a otros POM para que tenga versiones de biblioteca consistentes en todos sus proyectos.

Gradle 4.6 y versiones posteriores admiten la importación de listas de materiales, aunque en las versiones anteriores a Gradle 5.0 debe habilitarlo explícitamente agregando lo siguiente a su archivo **settings.gradle**:

enableFeaturePreview("IMPROVED_POM_SUPPORT")

Complementos comunes: Maven y Gradle comparten un enfoque común para extender la construcción a través de complementos. Aunque los sistemas de complementos son muy diferentes debajo de la superficie, comparten muchos complementos basados en características, tales como:

Shade/Shadow
Jetty
Checkstyle
JaCoCo
AntRun (see further down)

¿Por qué importa esto? Debido a que muchos complementos se basan en las convenciones estándar de Java, la migración es solo una cuestión de replicar la configuración del complemento Maven en Gradle. Como ejemplo, aquí hay una configuración simple de plugin de Maven Checkstyle:

```
...
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-checkstyle-plugin</artifactId>
  <version>2.17</version>
  <executions>
    <execution>
      <id>validate</id>
      <phase>validate</phase>
      <configuration>
        <configLocation>checkstyle.xml</configLocation>
        <encoding>UTF-8</encoding>
        <consoleOutput>true</consoleOutput>
        <failsOnError>true</failsOnError>
        <linkXRef>false</linkXRef>
      </configuration>
      <goals>
        <goal>check</goal>
      </goals>
    </execution>
  </executions>
</plugin>
...
```

Todo lo que se encuentre fuera del bloque de configuración se puede ignorar de forma segura al migrar a Gradle. En este caso, la configuración de Gradle correspondiente tiene el siguiente aspecto:

```
checkstyle {
    config = resources.text.fromFile('checkstyle.xml', 'UTF-8')
    showViolations = true
    ignoreFailures = false
}
```

Complementos que no necesitas: Vale la pena recordar que las compilaciones de Gradle suelen ser más fáciles de ampliar y personalizar que Maven. En este contexto, eso significa que puede que no necesite un complemento de Gradle para reemplazar uno de Maven.

Complementos poco comunes y personalizados: Puede encontrar complementos de Maven que no tienen contraparte en Gradle, especialmente si usted o alguien de su organización ha escrito un complemento personalizado. Tales casos dependen de que usted entienda cómo funciona Gradle (y potencialmente Maven), ya que generalmente tendrá que escribir su propio plugin.

7. Interfaz de línea de comandos

Ejecutar Gradle en la línea de comandos se ajusta a la siguiente estructura. Las opciones están permitidas antes y después de los nombres de las tareas.

gradle [taskName...] [--option-name...]

Si se especifican varias tareas, deben separarse con un espacio.

Las opciones que aceptan valores se pueden especificar con o sin **=** entre la opción y el argumento; Sin embargo, se recomienda su uso.

--console=plain

Las opciones que permiten el comportamiento de forma larga tienen la opción inversa especificada con **--no-**. Los siguientes son opuestos.

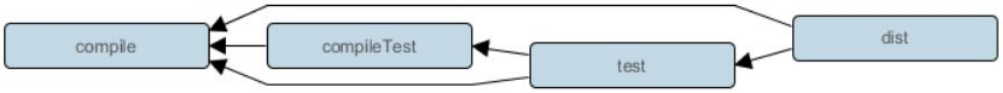
--build-cache
--no-build-cache

Muchas opciones de formato largo tienen equivalentes de opciones cortas. Los siguientes son equivalentes:

--help

-h

Puede excluir que una tarea se ejecute utilizando la opción **-x** o **--exclude-task** en la línea de comandos y proporcionar el nombre de la tarea para excluir.



gradle dist --exclude-task test

Puede forzar a Gradle a ejecutar todas las tareas ignoradas debido a las comprobaciones actualizadas usando la **--rerun-tasks**:

gradle test --rerun-tasks

De forma predeterminada, Gradle cancelará la ejecución y fallará la compilación tan pronto como falle alguna tarea. Para descubrir tantas fallas como sea posible en una sola ejecución de compilación, puede usar la opción **--continue**.

gradle test --continue

Tareas Comunes: Algunas tareas que utilizamos constantemente en cualquier proyecto Gradle son:

gradle build

gradle run

gradle check

gradle clean

gradle projects

gradle tasks

gradle tasks --all

8. Gradle Wrapper

La forma recomendada de ejecutar cualquier construcción de Gradle es con la ayuda de Gradle Wrapper (en resumen, solo "Wrapper"). The Wrapper es un script que invoca una versión declarada de Gradle, descargandola de

antemano si es necesario. Como resultado, los desarrolladores pueden comenzar a trabajar rápidamente con un proyecto Gradle sin tener que seguir procesos de instalación manuales, lo que le permite ahorrar tiempo y dinero a su empresa.

Beneficios:

- Estandariza un proyecto en una determinada versión de Gradle, lo que lleva a construcciones más confiables y robustas.

- Aprovisionar una nueva versión de Gradle para diferentes usuarios y entornos de ejecución (por ejemplo, IDEs o servidores de integración continua) es tan simple como cambiar la definición de Wrapper.

El archivo de propiedades de Wrapper generado **gradle/wrapper/gradle-wrapper.properties**, almacena la información sobre la distribución de Gradle.

- El servidor que aloja la distribución de Gradle.

- El tipo de distribución de Gradle. Por defecto, esa es la distribución **-bin** que contiene solo el tiempo de ejecución pero no muestra código y documentación.

- La versión de Gradle utilizada para ejecutar la compilación. De forma predeterminada, la tarea wrapper selecciona exactamente la misma versión de Gradle que se utilizó para generar los archivos Wrapper.

Un proyecto de Gradle generalmente proporciona un **build.gradle** y un archivo **settings.gradle**. Los archivos Wrapper se encuentran junto al directorio gradle y el directorio raíz del proyecto. La siguiente lista explica su propósito.

- **gradle-wrapper.jar**: El archivo JAR de Wrapper que contiene el código para descargar la distribución de Gradle.
- **gradle-wrapper.properties**: Un archivo de propiedades responsable de configurar el comportamiento de tiempo de ejecución de Wrapper, por ejemplo, la versión de Gradle compatible con esta versión.
- **gradlew, gradlew.bat**: Un script de shell y un script de proceso por lotes de Windows para ejecutar la creación con Wrapper.

9. Gradle con Java

Para crear una aplicación java utilizando Gradle utilizamos el siguiente comando:

gradle init --type <name>

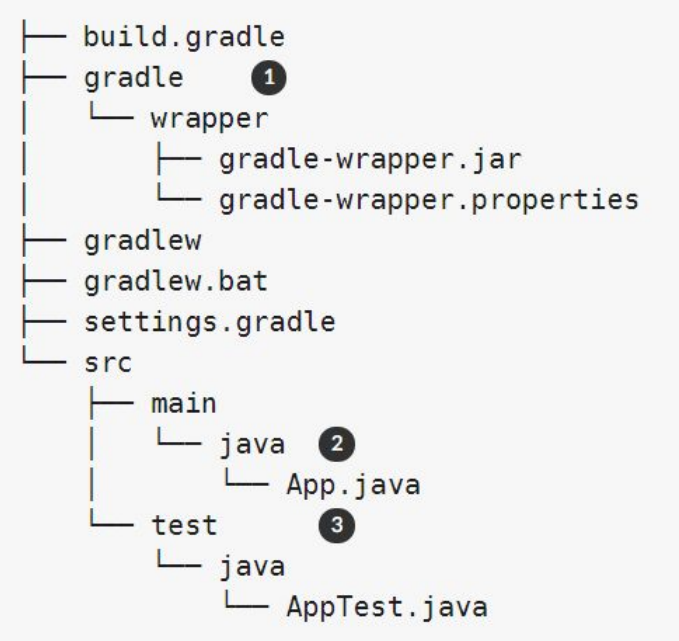
En donde name es uno de los siguientes:

- java-application
- java-library
- scala-library
- groovy-library
- basic

Esta guía usa el tipo java-application. El primer paso es crear una carpeta para el nuevo proyecto y ubicarnos en ella. Entramos a la línea de comandos y usamos el comando:

gradle init --type java-application

Esto nos genera un nuevo proyecto con la siguiente estructura:



1. Carpeta generada para archivos wrapper
2. Carpeta fuente predeterminada de Java
3. Carpeta de prueba predeterminada de Java

El archivo **settings.gradle** está muy comentado, pero solo tiene una línea activa: **rootProject.name='nombreDelDirectorioRaíz'**.

El archivo **build.gradle** generado también tiene muchos comentarios:

```
/*
 * This file was generated by the Gradle 'init' task.
 *
 * This generated file contains a sample Java project to get you started.
 * For more details take a look at the Java Quickstart chapter in the Gradle
 * user guide available at https://docs.gradle.org/4.5.1/userguide/tutorial_java_projects.html
 */

plugins {
    // Apply the java plugin to add support for Java
    id 'java'

    // Apply the application plugin to add support for building an application
    id 'application'
}

// Define the main class for the application
mainClassName = 'App'

dependencies {
    // This dependency is found on compile classpath of this component and consumers.
    compile 'com.google.guava:guava:23.0'

    // Use JUnit test framework
    testCompile 'junit:junit:4.12'
}

// In this section you declare where to find the dependencies of your project
repositories {
    // Use jcenter for resolving your dependencies.
    // You can declare any Maven/Ivy/file repository here.
    jcenter()
}
```

1. Repositorio de artefactos públicos de Bintray
2. Biblioteca de Google Guava
3. Biblioteca de pruebas JUnit
4. Clase con el método "main" (utilizado por el plugin application)

El archivo Build agrega el plugin **java** y **application**. El primero soporta los proyectos de Java y el último le permite designar una clase que contiene el método main, que puede ser ejecutado por la compilación desde la línea de comando. En la demostración, el nombre de la clase con el main es App.

Para construir el proyecto, ejecute el comando **build**. Puede usar el comando regular de gradle, pero cuando un proyecto incluye un script de wrapper, se considera una buena forma de usarlo.

gradlew build

La primera vez que ejecute **build**, Gradle comprobará si ya tiene las librerías Guava y JUnit en su caché. De lo contrario, las bibliotecas se descargarán y almacenarán allí. La próxima vez que ejecute la compilación, se usarán las versiones en caché. La tarea build compila las clases, ejecuta las pruebas y genera un informe de prueba. Puede ver el informe de prueba abriendo el archivo de salida HTML, ubicado en **build/reports/tests/test/index.html**.

Debido a que la compilación de Gradle utilizó el complemento de la aplicación, puede ejecutar la aplicación desde la línea de comando. Primero, use la tarea **tasks** para ver qué tarea ha sido agregada por el plugin.

La tarea **run** le dice a Gradle que ejecute el método main en la clase asignada a la propiedad **mainClassName**.

```
gradlew run
```

Resumen

Ahora tiene un nuevo proyecto Java que ha generado utilizando el plugin build init de Gradle. En el proceso, viste:

- Cómo generar una aplicación Java
- Cómo se estructuran el archivo de compilación generado y los archivos de muestra de Java
- Cómo ejecutar la compilación y ver el informe de prueba
- Cómo ejecutar una aplicación Java usando la tarea run desde el plugin application

Cómo Agregar una dependencia

En la sección **dependencies** se configura la dependencia. Algunas de las configuraciones estándar definidas por el plugin de la biblioteca de Java .

implementation: Las dependencias requeridas para compilar la fuente de producción del proyecto que no son parte de la API expuesta por el proyecto.

api: Las dependencias necesarias para compilar la fuente de producción del proyecto que forman parte de la API expuesta por el proyecto.

testImplementation: Las dependencias necesarias para compilar y ejecutar la fuente de prueba del proyecto.

Seguido agregamos **group**, **name** y **version.** Que son por decirlo la dirección donde se encuentra la dependencia, dicha información se utiliza luego para buscarla en un repositorio

Cómo agregar un repositorio

Un repositorio es una colección de módulos, organizada por **group**, **name** y **version**. Gradle comprende diferentes tipos de repositorios, como Maven e Ivy, y admite varias formas de acceder al repositorio a través de HTTP u otros protocolos.

Ubicamos la sección **repositories**, en esta sección se declara dónde encontrar las dependencias de tu proyecto por ejemplo:

```
repositories{
        jcenter()
}
```

También puede tener repositorios en el sistema de archivos local. Esto funciona tanto para repositorios de Maven como de Ivy.

```
repositories{
        maven{
                // url puede hacer referencia a una dirección local
                url "../local-repo"
        }
}
```

Un proyecto puede tener múltiples repositorios. Gradle buscará una dependencia en cada repositorio en el orden en que se especifiquen, deteniéndose en el primer repositorio que contenga el módulo solicitado.

Cibergrafía

La información para el curso fue tomada casi en su totalidad de la documentación oficial de Gradle.

https://gradle.org/

https://docs.gradle.org/current/userguide/userguide.html?_ga=2.160752701.1628691413.1532873854-1928928156.1519739528

https://gradle.org/releases/?_ga=2.168111009.1628691413.1532873854-1928928156.1519739528

https://gradle.org/maven-vs-gradle/

https://guides.gradle.org/migrating-from-maven/

https://docs.gradle.org/4.7/userguide/build_init_plugin.html?_ga=2.52585003.2003602173.1532976738-1928928156.1519739528#sec:pom_maven_conversion_

https://docs.gradle.org/current/userguide/gradle_wrapper.html

https://docs.gradle.org/current/userguide/tutorial_java_projects.html

https://guides.gradle.org/building-java-applications/?_ga=2.154115825.835635267.1533156752-1928928156.1519739528

https://docs.gradle.org/current/userguide/dependency_management_for_java_projects.html