Enter The Gradle

Ken Sipe - CTO, Gradleware

http://kensipe.blogspot.com/
http://del.icio.us/kensipe
twitter: @kensipe
ken.sipe@gradleware.com

Developer: Embedded, C++, Java, Groovy, Grails, C#, Objective C
Speaker: JavaOne 2009 Rock Star, NFJS, JAX
Microsoft MCP
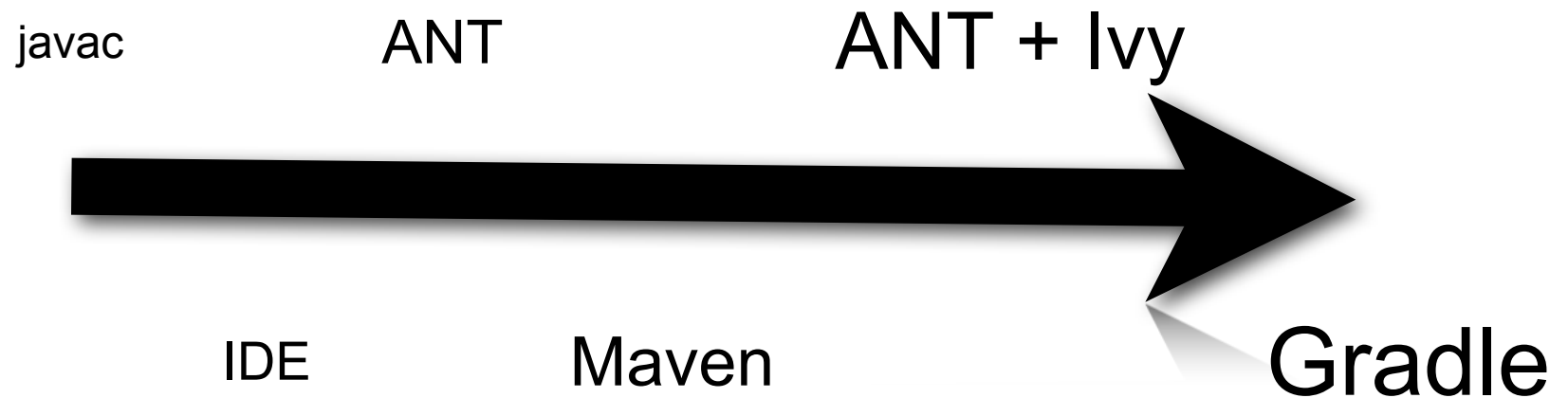Sun Certified Java 2 Architect
Master of Scrums
Agile Coach
Instructor: VisiBroker CORBA
           Rational Rose, OOAD

javac   ANT   ANT + Ivy

        IDE   Maven   Gradle

- Dependency management
- Versioning
- Compile Java code, build jars
    - + Other JVM Languages
- Execute tests and report results, fail build on failed tests
- Run quality-check tools (PMD, Findbugs, Checkstyles)
- File generation (XmlBeans, Xsl, Velocity, AspectJ)
- Property expansion / token substitution
- Build vs. deploy vs. release
- Full control when needed
- Cross-platform
- IDE Support
- Documentation / Support

- Cross Platform Builds
- IDE Independent
- XML "script" files
  - build.xml

- Defined Lifecycle
- Convention for Project Structure
- Plugins
- Dependency Management
- XML based
  - pom.xml

- Build steps defined and executed with Ant
- Dependencies managed with Ivy
- Ant targets to install, retrieve artifacts from Ivy repository

- hard to implement an algorithm in the build file;
  - simple if or for constructs are hard to achieve, and very unnatural
- hard to go beyond the foresight of the Ant/ Maven developers
- "build by convention" is not supported (Ant), or ties your hands because the configuration is hard (Maven),
- support for multi-module builds is limited
- boilerplate of XML is annoying

- Built on top of Ant + Ivy

- Build DSL written in Groovy

- Uses Groovy AntBuilder

  - ant.compile, ant.jar

- Plugins define common tasks to build different types of projects

  - java, groovy, war, …

Gradle is Declarative

Specify **what**...

...not **how**

Gradle is Declarative

... without being **Rigid**

- Many source dirs per project

- Dependencies per source dir

- JDK per source dir

- Many artifacts per project

http://gradle.org/

```
~$ gradle -t
:reportTask


------------------------------------------------------------

Root Project

------------------------------------------------------------

No tasks

BUILD SUCCESSFUL
```

NFJS
No Fluff Just Stuff.com

file:build.gradle
apply plugin: 'java'

~/projects/playground/gradle-nfjs$ gradle -t
:reportTask

Root Project

:assemble - Builds all Jar, War, Zip, and Tar archives.
   -> :jar
:build - Assembles and tests this project.
   -> :assemble, :check
:buildDependents - Assembles and tests this project and all projects that depend on it.
   -> :build
:buildNeeded - Assembles and tests this project and all projects it depends on.
   -> :build
:check - Runs all checks.
   -> :test
:classes - Assembles the main classes.
   -> :compileJava, :processResources
:clean - Deletes the build directory.
:compileJava - Compiles the main Java source.
:compileTestJava - Compiles the test Java source.
   -> :classes

```
createTask('hello') {  // depreciated
   println 'Hello World'
}

task hello << {
   println 'Hello world!'
}

task intro(dependsOn: hello) << {
   println "I'm Gradle"
}

project.tasks.add('someTask').doFirst {
   // do something
}
```

```
task hello << { println 'Hello' }
// direct API access is fine for single statements
hello.dependsOn otherTask
// for multiple access we prefer closure syntax
hello {
  onlyIf { day == 'monday' }
  dependsOn otherTask
}
// combining Configuration and Actions
task hello {
  onlyIf {
      day == 'monday'
  }
  doFirst {println 'Hello'}
}
```

```
apply plugin:'war'
version = 0.1

repositories {
    mavenCentral()
}

dependencies {
    compile "commons-lang:commons-lang:2.4"
}
```

```
apply plugin:'war'
version = 0.1
defaultTasks 'clean', 'build'

repositories {
    mavenCentral()
}

dependencies {
    compile "commons-lang:commons-lang:2.4"
}
```

Beyond the Basics

Does it **really** matter if your build system uses XML or Groovy?

Can there be aspects of the build that are difficult from a declarative perspective?

```
version = "1.0-${new Date().format('yyyyMMdd')}"

task sources {
   sourceSets.test.allGroovy
       .matching {include '**/*Demo*.groovy' }
       .files.each {
           println "$it.absolutePath"
       }
}
```

```
tasks.withType(Jar).allObjects { jarTask ->
  jarTask.osgi = new DefaultOsgiManifest()
  jarTask.doFirst { task ->
    importOsgiManifestIntoManifest(task) }
}
```

```
tasks.withType(Jar).allObjects { jarTask ->
    jarTask.manifest.mainAttributes(Provider: "CodeMentor Inc.")
}

tasks.withType(Compile).allObjects { compile ->
    compile.options.fork.executable = "$pathToJavac"
}

dependencies.allObjects { dependency ->
    throwExceptionIfDependencyIsGPL(dependency)
}
```

```
tasks.addRule("Pattern: ping<ID>") { String taskName ->
  if (taskName.startsWith("ping")) {
    task(taskName) << { // add task
      println "Pinging: " + (taskName - 'ping')
    }
  }
}
task groupPing(dependsOn: [pingServer1, pingServer2])
```

```
~/projects/playground$ gradle gP
:pingServer1
Pinging: Server1
:pingServer2
Pinging: Server2
:groupPing
```

Gradle Lifecycle

Initialization

- supports single and multi-project builds
- creates project instances for all that are taking part in the build

Configuration

- DAG (dependency acyclic graph) of tasks is created

Execution

- executed during initialization phase
- required for multi-project builds
  - in root project
- defines participating projects for builds

- optional for single-project build

settings.gradle

```
println 'executed during the init phase'
```

build.gradle

```
println 'executed during the config phase'

task test << {
    println 'executed during the execution phase'
}
```

```
task release(dependsOn: assemble) << {
    println 'We release now'
}

build.taskGraph.whenReady { taskGraph ->
    if (taskGraph.hasTask(':release')) {
        version = '1.0'
    } else {
        version = '1.0-SNAPSHOT'
    }
}
```

Gradle Dependencies

```
dependencies {
    runtime group: 'org.springframework', name: 'spring-core', version: '2.5'
    runtime 'org.springframework:spring-core:2.5', 'org.springframework:spring-aop:2.5'
}
```

```
dependencies {
        compile 'org.springframework:spring-webmvc:3.0.0.RELEASE'

        testCompile 'org.springframework:spring-test:3.0.0.RELEASE'
        testCompile 'junit:junit:4.7'
}
```

Options 1: Everything

```
configurations.compile.transitive = true

dependencies {
        compile 'org.springframework:spring-webmvc:3.0.0.RC2'

        testCompile 'org.springframework:spring-test:3.0.0.RC2'
        testCompile 'junit:junit:4.7'
}
```

Options 2: Selective

```
runtime('org.hibernate:hibernate:3.0.5') {
   transitive = true
}
runtime group: 'org.hibernate', name: 'hibernate', version: '3.0.5', transitive: true
runtime(group: 'org.hibernate', name: 'hibernate', version: '3.0.5') {
   transitive = true
}
```

```
dependencies {
    runtime files('libs/a.jar', 'libs/b.jar') runtime
    fileTree(dir: 'libs', includes: ['*.jar'])
}
```

```
List groovy = ["org.codehaus.groovy:groovy-all:1.5.4@jar",
"commons-cli:commons-cli:1.0@jar",
"org.apache.ant:ant:1.7.0@jar"]
List hibernate = ['org.hibernate:hibernate:3.0.5@jar',
'somegroup:someorg:1.0@jar']

dependencies {
   runtime groovy, hibernate
}
```

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

Or

```
repositories {
   mavenCentral name: 'single-jar-repo',
      urls: "http://repo.mycompany.com/jars"
   mavenCentral name: 'multi-jar-repos',
      urls: ["http://repo.mycompany.com/jars1", "http://repo.mycompany.com/jars1"]
}
```

Or

```
repositories {
   mavenRepo urls: "http://repo.mycompany.com/maven2"
}
```

```
repositories {
    flatDir name: 'localRepository',
    dirs: 'lib' flatDir dirs: ['lib1', 'lib2']
}
```

Custom Gradle

custom tasks and plugins

```
task hello(type: HelloTask)

task greeting(type: HelloTask) {
        greeting = 'greetings from new Task'
}

class HelloTask extends DefaultTask {
        def greeting = 'hello from HelloTask'

        @org.gradle.api.tasks.TaskAction
        def printGreeting() {
                println greeting
        }
}
```

- Plugins == Build Scripts
- Two Flavors:
  - ☐ Another build script (local or remote) (Script Plugin)
  - ☐ A class implementing org.gradle.api.Plugin (Binary Plugin)

Applying Plugins

- **Any gradle script can be a plugin.**
- **Binary plugins must be in the build script classpath**
  - can have id's (meta properties in the jar).
  - will learn later how to add elements to the build script classpath.
  - The build-in plugins are by default in the build script classpath.

```
apply from: 'otherScript.gradle'
apply from: 'http://mycomp.com/otherScript.gradle'
```

```
apply plugin: org.gradle.api.plugins.JavaPlugin
apply plugin: 'java'
```

Plugin-Id	applies
base	
java-base	base
groovy-base	java-base
groovy	groovy-base
scala-base	java-base
scala	scala-base
war	java
osgi	
code-quality	
maven	
eclipse	

Jetty Plugin Demo

with CamelCase

```
apply plugin: (GreetingPlugin)

class GreetingPlugin implements Plugin {

        def void use(Project project, ProjectPluginsContainer projectPluginsHandler) {
                project.task('hello') << {
                        println "Hello from the greetingPlugin"
                }
        }
}
```

** All projects using this plugin will now
   have the 'hello' task added and all its functionality

Common Interests

- **Jars can be added to the buildscript classpath**
  - ☐ Custom build logic
  - ☐ Plugins
  - ☐ Helper classes (e.g. commons-math)

```
buildscript {
  repositories { mavenCentral() }
  dependencies {
    classpath "commons-lang:commons-lang:3.1"
    classpath files('lib/foo.jar')
  }
}
```

```
war.doLast {
  ant.unzip(src: war.archivePath,
   dest: "$buildDir/exploded")
}
```

NFJS

build.xml

```
<project> <target name="hello" depends="intro">
<echo>Hello, from Ant</echo> </target>
</project>
```

build.gradle

```
ant.importBuild 'build.xml'

hello.doFirst { println 'Here comes Ant' }
task intro << { println 'Hello, from Gradle'}
```

```
output:
~/projects/playground/gradle/ant$ gradle hello
:intro
Hello, from Gradle
:hello
Here comes Ant
[ant:echo] Hello, from Ant
```

■Integration with Maven repositories

- ☐autogeneration of pom.xml

- ☐install to local Maven repo

- ☐deploy to any remote Repo

- ☐full maven metadata generation

■Integration of Maven builds in the future

```
ant.java(classname: 'com.my.classname', fork: true,
        classpath: "${sourceSets.main.runtimeClasspath.asPath}")
```

```
apply plugin:'java'

def cobSerFile="${project.buildDir}/cobertura.ser"
def srcOriginal="${sourceSets.main.classesDir}"
def srcCopy="${srcOriginal}-copy"

repositories {
    mavenCentral()
}

dependencies {
        testRuntime 'net.sourceforge.cobertura:cobertura:1.9.3'
        testCompile 'junit:junit:4.5'
}

test.doFirst  {
    ant {
        delete(file:cobSerFile, failonerror:false)
        delete(dir: srcCopy, failonerror:false)
        taskdef(resource:'tasks.properties', classpath: configurations.testRuntime.asPath)
        copy(todir: srcCopy) {
            fileset(dir: srcOriginal)
        }

        'cobertura-instrument'(datafile:cobSerFile) {
            fileset(dir: srcOriginal,
                    includes:"my/classes/**/*.class",
                    excludes:"**/*Test.class")
        }
    }
}
```

```
test {
    // pass information on cobertura datafile to your testing framework
    // see information below this code snippet
}

test.doLast {
    if (new File(srcCopy).exists()) {
        // replace instrumented classes with backup copy again
        ant {
            delete(file: srcOriginal)
            move(file: srcCopy,
                    tofile: srcOriginal)
        }
        // create cobertura reports
        ant.'cobertura-report'(destdir:"${project.buildDirName}/test-results",
            format:'html', srcdir:"src/main/java", datafile:cobSerFile)
    }
}
```

```
// File: build.gradle
loadConfiguration()

task printProps << {
    println "serverName:  $config.serverName"
    println "mail.server: $config.mail.server"
    println "mail.port:   $config.mail.port"
}

def loadConfiguration() {
    def environment = hasProperty('env') ? env : 'dev'
    setProperty 'environment', environment
    println "Environment is set to $environment"

    def configFile = file('config.groovy')
    def config = new ConfigSlurper(environment).parse(configFile.toURL())
    setProperty 'config', config
}
```

```
// File: config.groovy
mail {
    server = 'localhost'
    port = 25
}

environments {
    dev {
        serverName = 'http://localhost:9090'
    }

    test {
        serverName = 'http://testserver'
        mail {
            server = 'mail.testserver'
        }
    }

    prod {
        serverName = 'http://www.nfjs.com'
        mail {
            port = 552
            server = 'mail.host.com'
        }
    }
}
```

~/projects/playground/gradle/env$ gradle -q -Penv=test pP
Environment is set to test
serverName: http://testserver
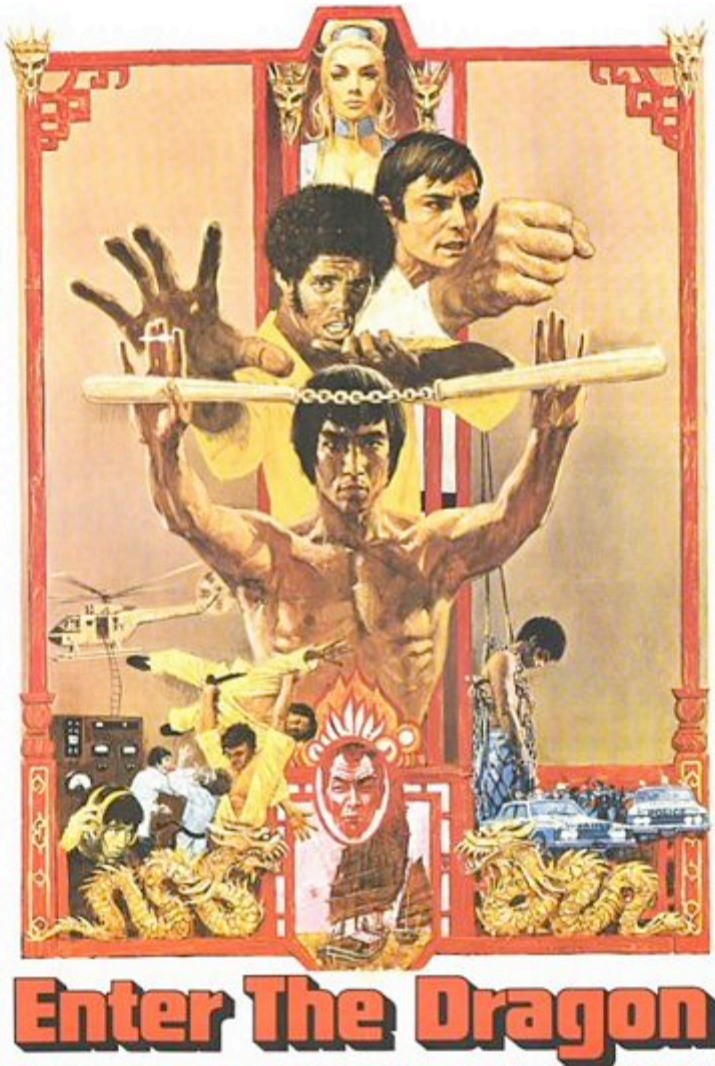mail.server: mail.testserver
mail.port: 25

thanks to mrhaki for the tip!

```
task wrapper(type: Wrapper) {
    gradleVersion = '0.8'
}
```

task execution results:

build
build.gradle
gradle-wrapper.jar
gradle-wrapper.properties
gradlew.bat
gradlew

•Gradle is version 1.0-Milestone2a!

but

•It is very powerful!

■ Closing and Q&A

☐ Please fill out the session evaluation

☐ Ken Sipe

- ■ <u>ken.sipe@gradleware.com</u>
- ■ kensipe.blogspot.com
- ■ twitter: @kensipe