Maven, not just another build tool

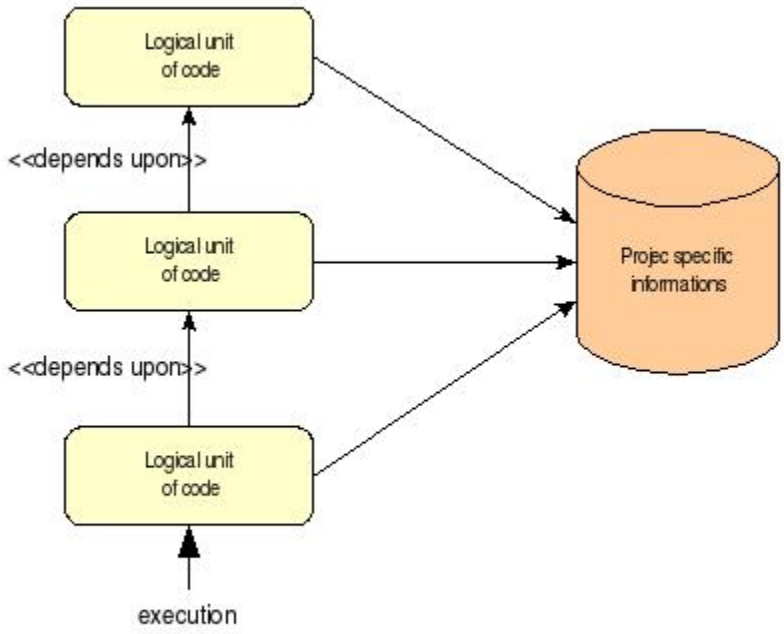
Maven, not just another build tool

Tradional build tools (make, Ant, ...) are usually nothing more than some specialized script languages which are by definition very task-oriented. A functional build script can usually be divided into three kinds of content:

1. Some execution code logically grouped into units (compiling, jar packaging, generate javadoc, ...).
2. The interrelationship between the different units of code (execution order).
3. The project specific informations (file locations, libraries, ...).

Those who have used Apache Ant (the most popular Java build tool) in the past probably feel familiar with these elements. In fact, Ant's three main concepts are directly mapped from those:

1. Tasks are reusable logical units of some execution code.
2. Targets specify which other tasks should be executed before a given task.
3. Properties define project specific informations.

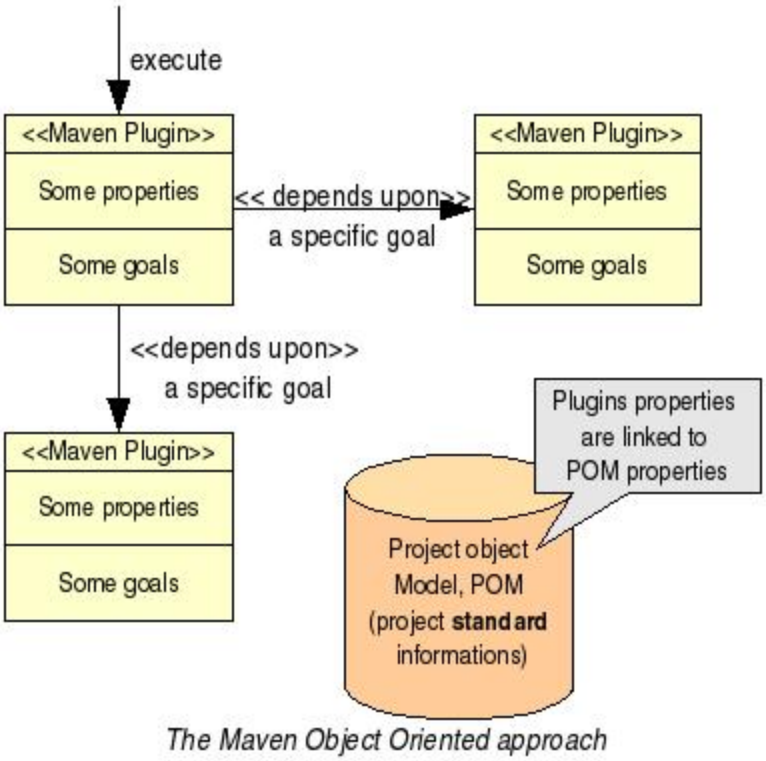


The traditional build script approach

This approach is very similar to the procedural programming approach. The build script is just a collection of different functions separated from the data they depend on and executed in a given order. While procedural approaches allow some form of modularity and therefore some form of reuse, more sophisticated forms are possible with an object oriented approach.

Enter the world of Maven

Maven brings the object oriented approach to the build tools table. Object oriented approach states that a program is usually made of several individual units, the *objects*, capable of some processing and that interact with each other by sending messages. Of course, you already know that since you are a Java programmer. Maven *objects* are known as plugins. They encapsulate some behaviour (goals) and some data (properties) under a logical unit, namely the plugin. Each Maven-enabled project just needs to declare what goals it wants to use, the different property values and optionally in which order the goals should execute (we'll see how it is done in the next lessons). It is important to remember that the object oriented approach still needs the same information as the procedural approach. The difference is that the object oriented approach organizes the information in a very different way than the procedural approach - one that gives more flexibility and ease applying changes.



The Maven Object Oriented approach

About Ant
Some readers at this point may think Ant can be thrown out the window when you commit to Maven as being your project build tool. The truth is far from that, Ant can still be a fundamental part of your build if you decide so because Maven plugins, as you will see in more advanced trails, can be written using Java or Ant. This way you limit your Ant use to what a lof of people see as its true strength, a powerful task framework while letting Maven orchestrate your project build in a OO fashion.

Build patterns

Maven 1 has proven the Object Oriented approach is indeed quite powerful but there were still some shortcomings not solved. For instance, as you can see in the previous picture, most of the plugin goals are dependent upon the pre-execution of some other plugin's goal. In order to make your project build successfully, you have to make sure the goal's dependency graph is correct. The same problem is also present with task-oriented build tools (I can't count how many times I have tried to figure out in a weird Ant script which target is doing what). It isn't that much of a problem on small projects and even on bigger projects for the original project developers. The problem usually appears when a new developer joins the project team. If no conventions are used, it can take him a significant amount of time to become comfortable with the project build - time which would have been better spent developing the application.

Maven 2 addresses this problem elegantly by promoting build patterns. According to Martin Fowler in his famous book Patterns of enterprise application architecture *—The focus of the pattern is a particular solution, one that's both common and effective in dealing with one or more recurring problems*. Hence, a pattern is just a formalized and consistent way of describing some best practices in a domain. It is important to understand it can't be directly translated into code or some other physical implementation. According to the design patterns Bible, Design Pattern - Elements of Reusable Object-Oriented Software; *—The solution doesn't describe a particular concrete design or implementation, because a pattern is like a template that can be applied in many different situations.* Build patterns typically show the relationships between the different build tasks (which task should be completed before activating the current one) and the project's physical structure, without specifying the tasks composing the project build and the different physical components the project is made of.

While project builds may seem very different at first and therefore finding some patterns may seem impossible, there are some general solutions that can be observed. In the case we are interested in, we can observe a project build is usually divided into distinct finite phases (compiling, testing, packaging, deploying, ...) This pattern is known as the build lifecycle pattern and is covered in detail in the The Build Lifecycle lesson. It allows you to manage your plugin goals dependency graph easily in a very consistent manner. The build lifecycle is one of the main patterns around which Maven 2 is built but there are others. Along this tutorial, in the appropriate lessons, specific build patterns and their respective Maven implementations will be discussed.

More than a build tool

It is also important to understand that **Maven is more than just a build tool**. While Maven can manage your project build effectively, it can also

generate a project Web site, manage your dependencies, generate specific reports, ... In concrete terms, Maven will do whatever you ask it to, if you have the correct plugin installed, using the information provided in the project POM.