**Assignment 15.2**

**Maven:**

Maven is a "build management tool", it is for defining how your .java files get compiled to .class, packaged into .jar (or .war or .ear) files, (pre/post)processed with tools, managing your CLASSPATH, and all others sorts of tasks that are required to build your project. It is similar to [Apache Ant](http://ant.apache.org/) or [Gradle](http://www.gradle.org/) or [Makefiles](http://en.wikipedia.org/wiki/Makefiles) in C/C++, but it attempts to be completely self-contained in it that you shouldn't need any additional tools or scripts by incorporating other common tasks like downloading & installing necessary libraries etc.

It is also designed around the "build portability" theme, so that you don't get issues as having the same code with the same buildscript working on one computer but not on another one (this is a known issue, we have VMs of Windows 98 machines since we couldn't get some of our Delphi applications compiling anywhere else). Because of this, it is also the best way to work on a project between people who use different IDEs since IDE-generated Ant scripts are hard to import into other IDEs, but all IDEs nowadays understand and support Maven ([IntelliJ](http://www.jetbrains.com/idea/features/ant_maven.html), [Eclipse](http://www.eclipse.org/m2e/), and [NetBeans](http://wiki.netbeans.org/Maven)). Even if you don't end up liking Maven, it ends up being the point of reference for all other modern builds tools.

Maven continues using XML as the format to write build specification. However, structure is diametrically different. While Ant requires developers to write all the commands that lead to the successful execution of some task, Maven relies on conventions and provides the available targets (goals) that can be invoked. As the additional, and probably most important addition, Maven introduced the ability to download dependencies over the network (later on adopted by Ant through Ivy). That in itself revolutionized the way we deliver software.

However, Maven has its own problems. Dependencies management does not handle well conflicts between different versions of the same library (something Ivy is much better at). XML as the build configuration format is strictly structured and highly standardized. Customization of targets (goals) is hard. Since Maven is focused mostly on dependency management, complex, customized build scripts are actually harder to write in Maven than in Ant.

Maven configuration written in XML continuous being big and cumbersome. On bigger projects it can have hundreds of lines of code without actually doing anything “extraordinary”.

Main benefit from Maven is its life-cycle. As long as the project is based on certain standards, with Maven one can pass through the whole life cycle with relative ease. This comes at a cost of flexibility

**GRADLE:**

Gradle is an automated project building tool that uses the concepts of both Apache Ant and Apache Maven but is based on a domain specific language rather than the traditional XML approach. Gradle is designed to support multi-project builds that are quite large and are difficult to manage via Ant or Maven.

This article will discuss the concepts of Gradle as a project building tool and also show how to configure and build a sample Java project.

* **Declarative build**- Gradle provides declarative language elements that we can be assembled as per our choice. This declarative language is extensible which enables us to add our own new language or enhance the existing one. These elements also provide build by convention support for Java, Groovy and other modern Java based technologies.
* **Language for dependency based programming**- The declarative language is very flexible and enables Gradle to support the specific requirements.
* **Structure the build**- Gradle allows us to apply common design principles to our build, which enables us to create a structured build for our applications.
* **Scalability**- Gradle has the ability to easily scale from a simple single project build to a huge enterprise multi-project build. It enables the incremental build and also has the ability to tackle the performance issues that plague large enterprise build scripts.
* **Multi-project support**- Gradle supports multi-project builds. It enables us to maintain the relationships between different projects in the case of a multi-project build environment. It also supports partial builds. We can build a single subproject out of an enterprise application. While building the single subproject, Gradle takes care of the other subprojects if the said subproject has dependency on other subprojects.
* **Dependency Management**- Gradle provides different ways to manage internal as well as external dependencies. It provides supports for all kinds of dependency management, starting from transitive dependency management with remote access to Maven or any other repository-- even the local file system.
* **Integration tool**- Gradle can easily import any Ant project and its targets and converts them into native Gradle tasks at runtime. Gradle also provides conversion mechanism to convert the maven pom.xml file into Gradle script.
* **Migration**- Gradle easily adapts to any structure. We can easily develop the Gradle build in the same production branch.
* **Groovy Support** - Gradle scripts are written in Groovy and not XML based.

**SBT**

**sbt** is an [open source](https://en.wikipedia.org/wiki/Open_source) [build tool](https://en.wikipedia.org/wiki/Build_tool) for [Scala](https://en.wikipedia.org/wiki/Scala_(programming_language)) and [Java](https://en.wikipedia.org/wiki/Java_(programming_language)) projects, similar to Java's [Maven](https://en.wikipedia.org/wiki/Apache_Maven) or [Ant](https://en.wikipedia.org/wiki/Apache_Ant).

Its main features are:

* native support for compiling Scala code and integrating with many Scala [test frameworks](https://en.wikipedia.org/wiki/Test_automation)
* build descriptions written in Scala using a [DSL](https://en.wikipedia.org/wiki/Domain_Specific_Language)
* dependency management using [Ivy](https://en.wikipedia.org/wiki/Apache_Ivy) (which supports Maven-format repositories)
* Continuous compilation, testing, and deployment.
* Integration with the Scala interpreter for rapid iteration and debugging.
* Support for mixed Java/Scala projects.