# Clase 12/3

#PAW

- [ ] Configurar tabs, indentaciones

---

## Maven

N-capas independientes y auto-contenida. Ordenadas. Cada una sólo le puede hablar a la de abajo. Esto permite encapsular comportamiento.

Cada capa separar en un proyecto aparte. Cada una podría tener sus propias dependencias.

Hace pocas cosas: resolver artefactos, descargarlos y ejecutarlos.

**Maven home:** ~*m2*repository

- [ ] **NO SUBIR ARCHIVOS AUTOGENERADOS (.idea) ( de intellij de eclipse, etc...)**
➜ .gitignore

$ mvn archetype:generate
$ root
$ 4 ➜ pom-root for multi model projects
Coordenadas que definen proyecto:
**groupId**: identificador de la org que crea el artefacto : ar.edu.itba.paw
*groupId*:*nombre del artefacto version*

**artifactId**: webapp2018a

**versión**: 1.0-SNAPSHOT ➜ snapshot es convención para "esto es version de desarrollo", puedo cambiarlo. Se van acumulando.

package: *enter para el defecto*

$ tree

generó pom : description de mi proyecto

---

$ mvn archetype:generate
$ maven-archetype-webapp ➜ genera esqueleto de aplicación web

este solo tiene .jsp (lenguaje de templating) no tiene códigos

todo lo que está afuera de WEB-INF es público

---

$ mvn archetype:generate
$ maven-archetype-quickstart

---

en mi pom padre configuro cosas que los hijos no quiero que repitan

---

$ mvn eclipse:clean (para el plugin eclipse)

---

IP 0.0.0.0.8080 todas las interfaces
Error de compilación le informamos a maven explícitamente el plugin de compilación de java. Podemos definir properties para las versiones de todo, para evitar cabrear cosas y que impacte en todos lados.

<maven.compiler.source>1.8
<maven.compiler.target>1.8

---

## MVC

El Model debería funcionar con cualquier V [JavaFX, etc].

---

## Inversion of Control

*Le pasas el Motor al Auto.*

Mucho más fácil de configurar. Es transparente cambiar Motores y sigue andando.

Debemos programar contra **interfaces**.

Puedo hacer que si no programo con interfaces, no compila.

Querer transformar error de Runtime en error de Compilación.

---

## Pattern Front Controller

Provisto por Spring.

Se encarga de dirigir los requests sin implementar Servlet.

---

## Pattern View Strategy

ViewResolver

---

## Elegir Annotations antes que agregar tags a xml

---

## @Controller

@RequestMapping("/")
public ModelAndView helloWorld() {
    final ModelAndView man = new ModelAndView("index"); *pasale estos datos a esta view*. **final** ➜ deseable reducir mutabilidad
    mav.addObject("greeting", "PAW");
    return mav;
}

---

## XSiteScripting

mav.addObject("greeting", "<script>alert(chan)</script>");

**Nunca jamás en la vida:** ${greeting}! o con el # =

siempre <c: out value="${greeting}" escapeXml="true"/>

---

## Capas

En este orden:

- *WEBAPP*: HTML, CSS, JS, controllers
- *SERVICIOS*:  Ofrece un servicio al sistema. Pueden ser 2 cosas: acceso a (..) (mails, por ejemplo), o un patrón (GoF): **Facade** me ofrece una interfaz simple para un proceso complejo. i.e.: register(user) ➜ implica muchos pasos en el proceso de registrar. Hacia la capa de arriba solo expongo 'register'.
- *PERSISTENCIA*: convertir tablas base de datos. orientado a mi aplicación; no es un framework. Debería ser una *Single Source of Truth*: no hay forma de instancias modelo sin esta capa. Esta capa deber conocer y manipular el modelo. Conecta con DB.
- *MODELO*: mis entidades, tablas, sus atributos, valida passwords, …

  Todas menos Modelo tienen *INTERFACES*.