Maven

* Build Tool.
* Project management tool.

- Common problems and activities

i) Multiple jars.

ii) Dependencies and Versions.

iii) Project Structure.

iv) Building, publishing and deploying.

- Download the Maven In System (Using Linux).

Step 1: Download zip File from browser

Step2: Export File in your system

Step 3: Copy File into your home directory

Command:

```
export M2_HOME = /Path for your downloaded File
```

Step4: Add ~~to~~ the Maven ~~ito~~ into Env Variables

Command

```
export PATH = / Path for bin Folder : ${PATH}.
```

Step 5: Check it's Working or NOT

Command

```
mvn --version
```

* Archetype (Blueprint for new Maven Project)

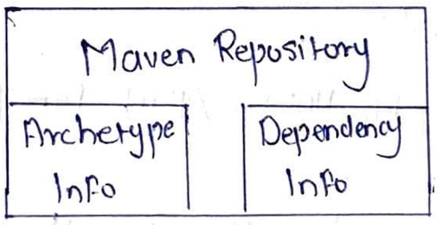
- A Maven Archetype is a project template used to generate a standardized Maven project with predefined files, folders and configration.

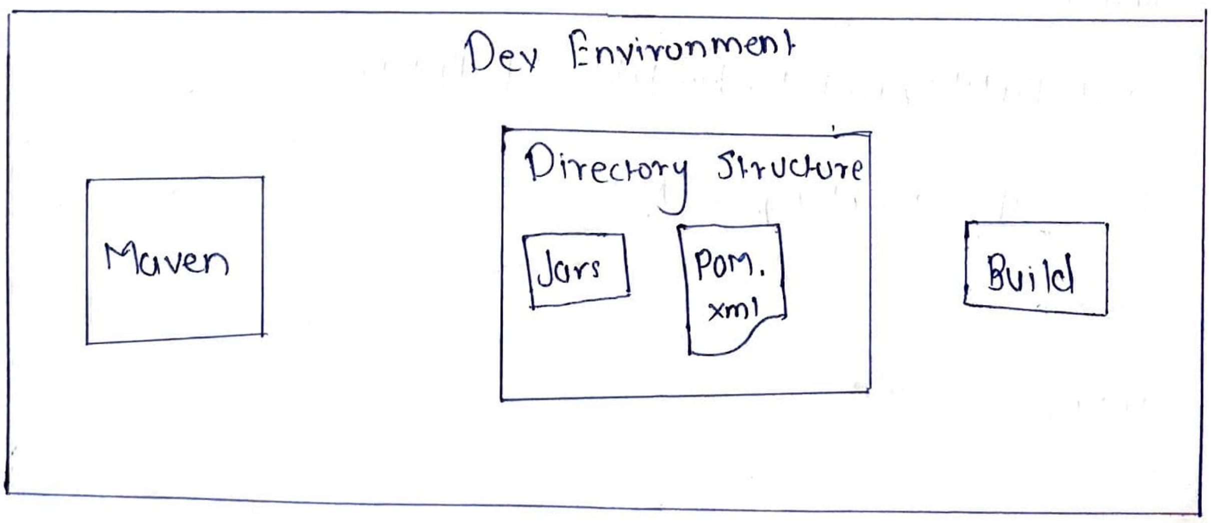
* Compile and Run first Maven Program.

Commands.

```
mvn compile
mvn package
java -cp target/MavenTestApp-1.0-SNAPSHOT.jar org.sachitp.dev.App.
```

*How it Works





* Some Phases

i) validate

ii) compile

iii) test

iv) package

v) install

vi) deploy.

* Adding a Dependencies.

A dependency is an external library (JAR) that your project needs to compile, run, or test and which Maven automatically downloads and manages for you.

- Where to add

pom. xml.

```
<dependencies>
    <dependency>
        <!-- add your dcode>
        <! ~ available on internet>
    </dependency>
</dependencies>
```

* What Maven did ?

i) Project Template

ii) Build

* Maven archetype

i) Folder Structure

ii) pom.xml ⟶ (What it consists) i) Maven co-ordinates

↕ ii) Metadata

- archetype : generate iii) Build information

iv) Resources and dependencies

i) Archetype

ii) Group ID

iii) Artifact ID

iv) Version

v) Package

* Maven Build

i) Build lifecycle

ii) Consits of phases → Compile
↳ Test
↳ Pakage

iii) Default behaviour of phases.

iv) Specify the build phase you need. Previous phase automatically run.

* Plugins

A plugin is a software component that adds extra features or behaviour to an existing application without changing its core code.

Note: generics :- Generics allows you to define classes, interfaces, and methods with a placeholder for data types which filled in when you use them.

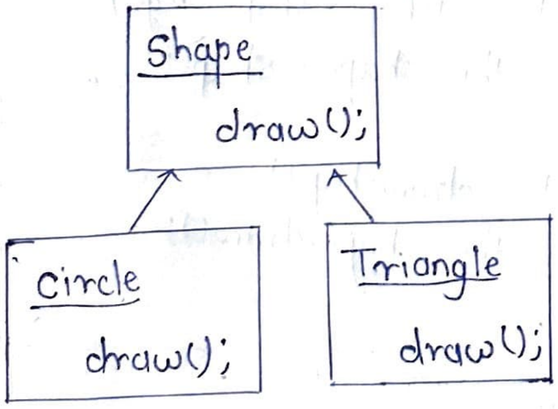
Spring

* Dependency Injection
* Understand with example

Circle
draw();
Triangle
draw();

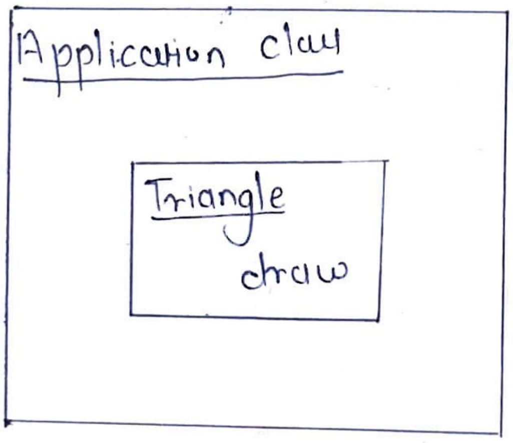
```
Application class
    Triangle myTriangle = new Triangle();
    myTriangle.draw();

    Circle myCircle = new Circle();
    myCircle.draw();
```

- Using polymorphism



```
Application Case

    Shape shape = new Triangle();
    shape.draw();

    Shape shape = new Circle();
    shape.draw();
```

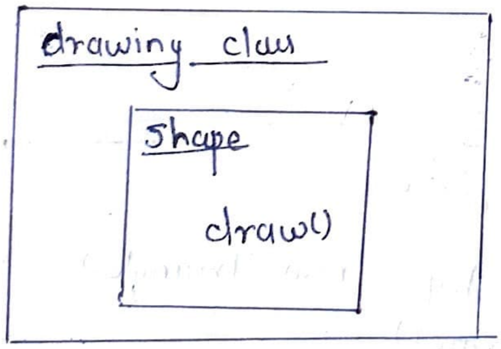
* Method Parameter.



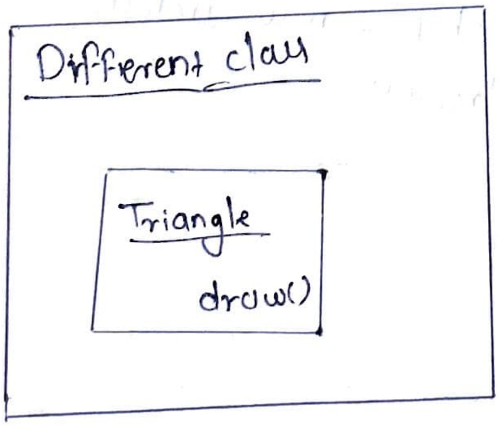
```
Application class
public void myDrawMethod(shape shape){
        shape.draw();
}
```

```
Somewhere else in the class

    Shape shape = new Triangle();
    myDrawMethod(shape);
```

* Class member Variable



```
Drawing class
    protected class Drawing{
        private shape shape;
        public setshape(Shape shape){
              this. shape = shape;
        }
        public drawShap(){
              this. shape.draw();
        }
```



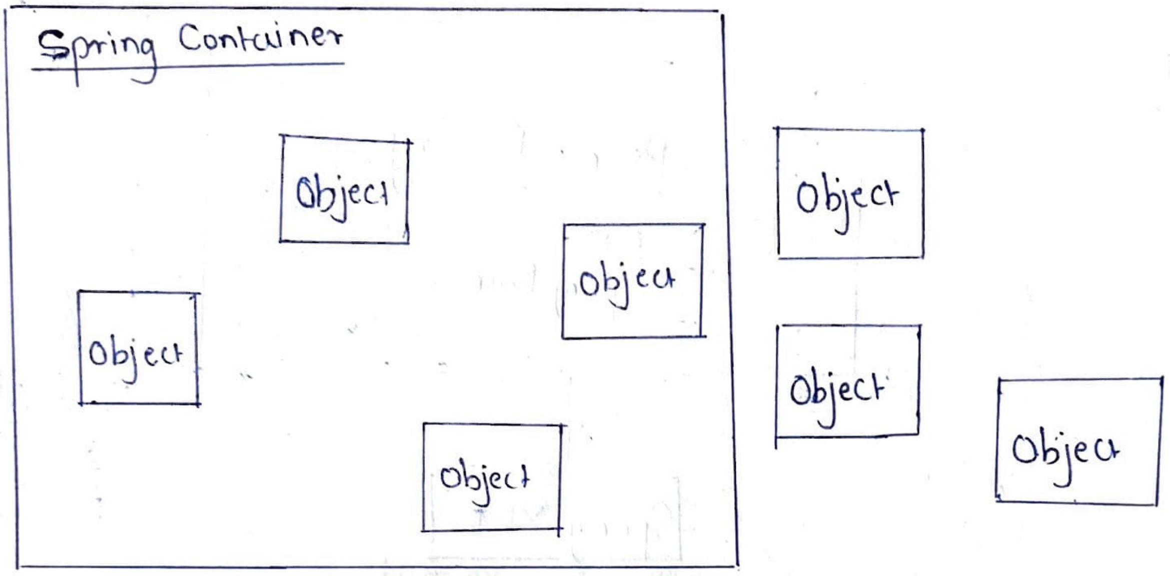
```
Different class

Triangle myTriangle = new Triangle();
drawing.setShape(MyTriangle);
drawing.drawShape();
```

* Defination for Dependency Injection

Dependency Injection promots lorre coupling by supplying required objects to a claw from outside insted of claw creating them.

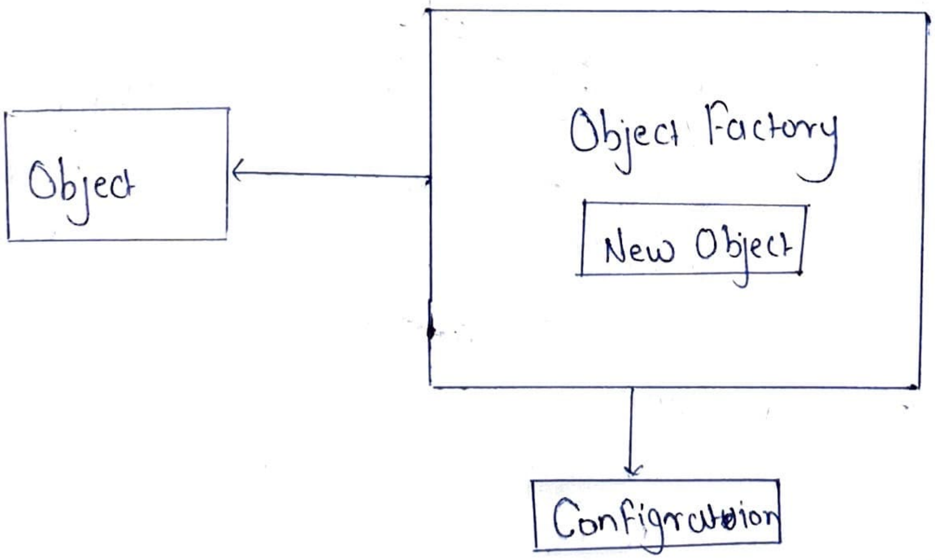
* Spring Factory Bean

- A spring Container



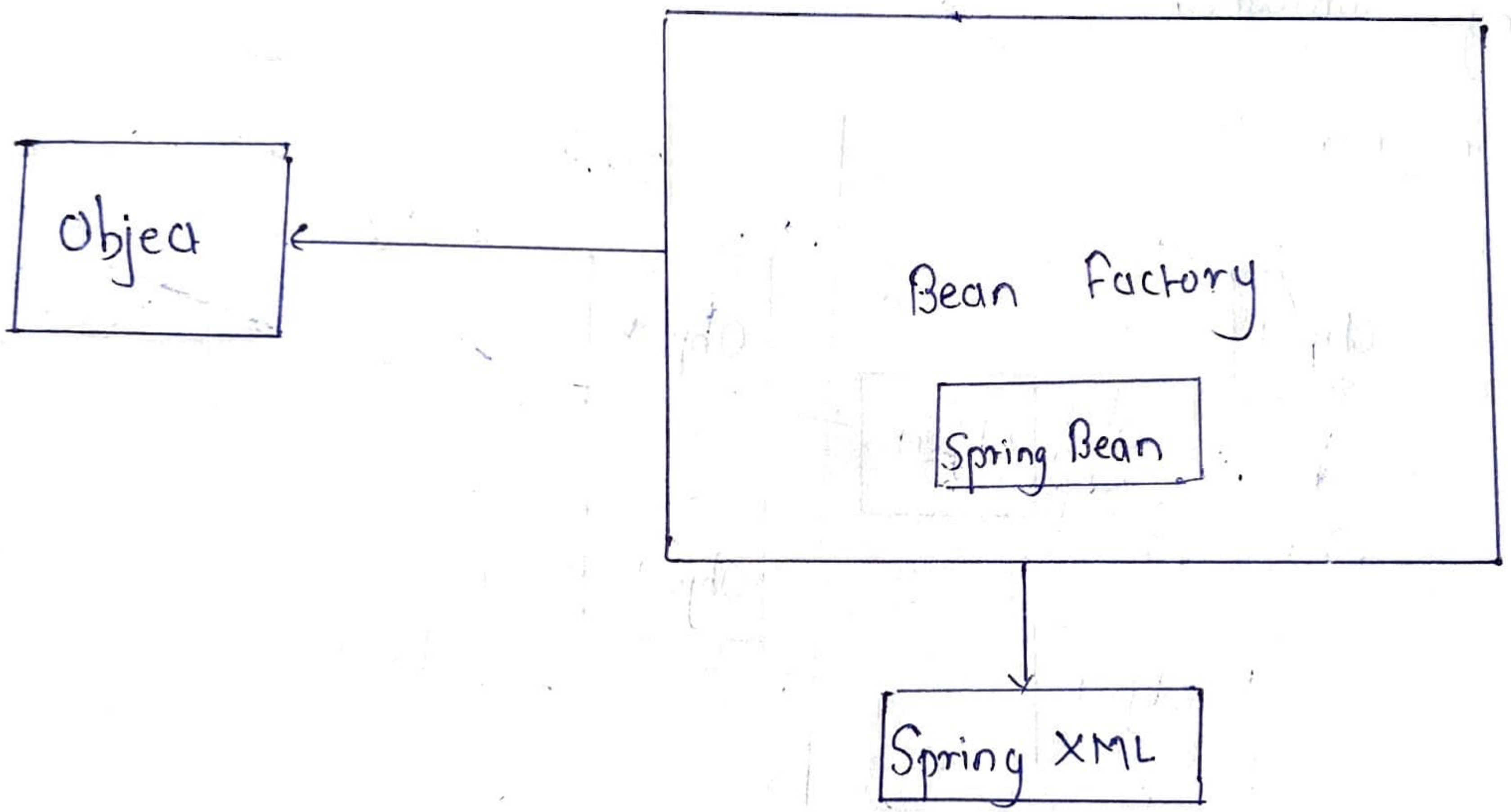
Spring Container is a runtime environment that manages object and their dependencies using Inversion of control (IOC) and Dependency Injection (DI).

- Factory Pattern



Factory pattern provides an interface for creating objects, but lets subclass or a factory class decid which class to initiate.

- Spring Bean Factory.



* AutoWiraing

Autowiring in spring is a mechanism where the spring IoC container automatically Injects required dependencies into a bean, insted of you manually wiring them.

* Basic Bean Scope.

In Spring, a bean scope defines how long a bean lives and how many instances of that bean spring creates inside the container.

① Singleton Scope (DEFAULT)

i) Only One instance of the bean is created per Spring Container

ii) Same object is shared Everywhere.

② Prototype Scope (other Basic Scope).

1) New instance every time bean is requested.

* Web - Aware Scope in Spring.

Web-aware Scope are spring bean scope that exist only in web application (Servlet - based appr). They depend on HTTP. request, session, or application context.

① request Scope
i) One bean instance per HTTP request.
ii) New object for every request.
iii) Destroyed when request ends.

② Session Scope.
i) One bean instance per HTTP session.
ii) Same object reused for same user session.

③ application Scope.
i) One bean per web application
ii) shared across all users ~~of~~ and sessions.

④ Websocket Scope. (Advanced)
i) One bean per web socket session.

* ApplicationContextAware.

ApplicationContextAware is a spring callback interface that allows a bean to get access to the spring ApplicationContext-object at runtime.

* Bean Name Aware

BeanNameAware is a spring Aware interface that allows a bean to know the name by which it is regestried inside the spring container.

* Bean Defination Inheritance.

Bean Defination Inheritance allows one spring bean defination (child) to inherit configuration metadata from another bean defination (parent).

* Bean Post Processor.

BeanPost Processor is a spring extension point that allows you to intercept and modify beans before and after their initialization.

* Bean Factory Post Processor.

A Bean Factory Post Processor is a spring extension point that allows you to modify bean definitions (metadata) before any bean object is created.

* Annotations

An annotation is a special form of metadata in Java (and many other language) that provides extra information about code without changing how the code itself works

Think of annotations as labels or instructions that tell the compiler, framework, or runtime how to treat a class, method variable, or field

1) @Autowired :- i) It is used for Dependency Injection (DI)

ii) It tells the spring framework container to automatically injects a required object (bean) into another bean.

2) @Required :- i) It is a spring annotation used to mark a setter method as mandatory.

ii) If the required property is not injected, the spring framework container throws an exception at startup.

3) @Resource :- i) It is a dependency injection annotation from Java (JSR-250) that spring also supports. It injects a bean primarily by name, not by type.

4) @Post Construct :-

i) It is a lifecycle annotation used to run a method after a bean is created and all its dependencies are injected, but before the application start serving request.

5) @Pre Destroy :-

i) It is a lifecycle annotation used to run a method just before a spring bean is destroyed. typically when the application context is shutting down.

⑥ @Component :-

i) It is a stereotype annotation that ~~makes~~ marks a class as a spring-managed ~~them~~ bean.
ii) When spring perform component scanning, it automatically detects creates, and manages object of classes annotated with @component

7) @Service :-

i) It is a sterotype annotation used to ~~make~~ mark a class as a service-layer component that contains business logic
ii) It is a specialized form of @component, meaning, meaning spring will automatically detect it during component scanning and reg it ~~has~~ as a bean.

8) **@Repository** :-

i) It is a Stereotype annotation used to mark a class as a Data Access Object (DAO) or persistance layer Component.

9) **@Controller** : It is a stereotype annotation used in Spring MVC to mark a class as a web controller that handles HTTP request and returns view (UI Pages)