1. Servlet:

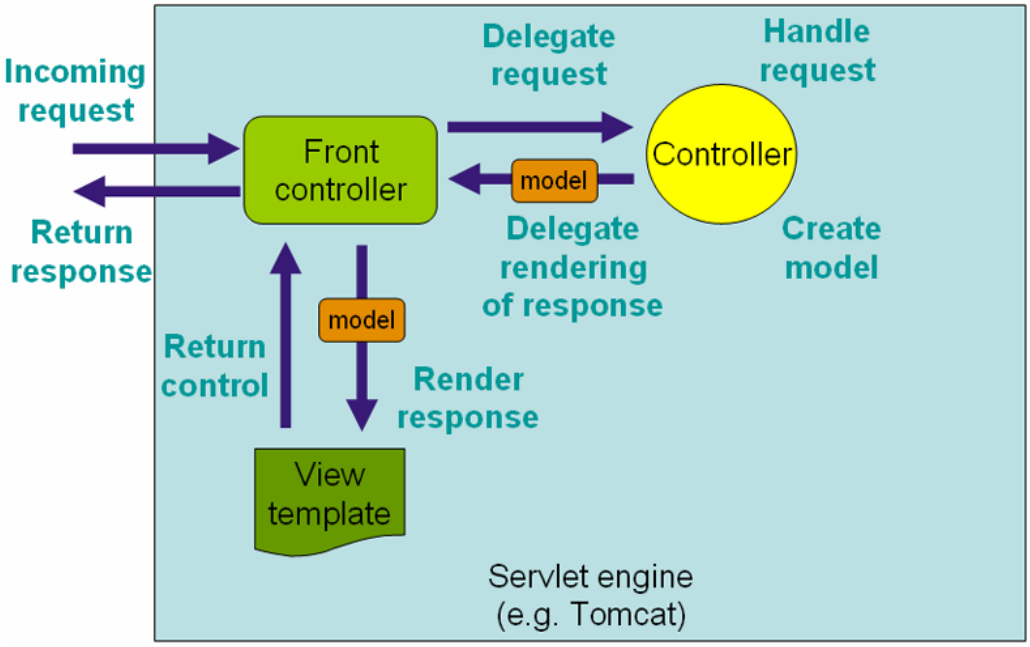
- A servlet is a **small Java program** that runs within a Web server. Servlets **receive and respond to requests from Web clients**, usually across HTTP, the HyperText Transfer Protocol.

- **Java Server Pages (JSP)** is a technology which is used to **develop web pages by inserting Java code into the HTML pages** by making special JSP tags. The JSP tags which allow java code to be included into it are **<% ----java code(Scriptlet)----%>.**

- **The Expression Language (EL)** simplifies the **accessibility of data stored** in the **Java Bean** component, and other objects like **request, session, application** etc.
Ex : **${ expression }**

Note: Don't use Scriptlet in JSP(because in jsp only view related must be present, not business logic) instead use EL to display data.

2.Spring mvc:



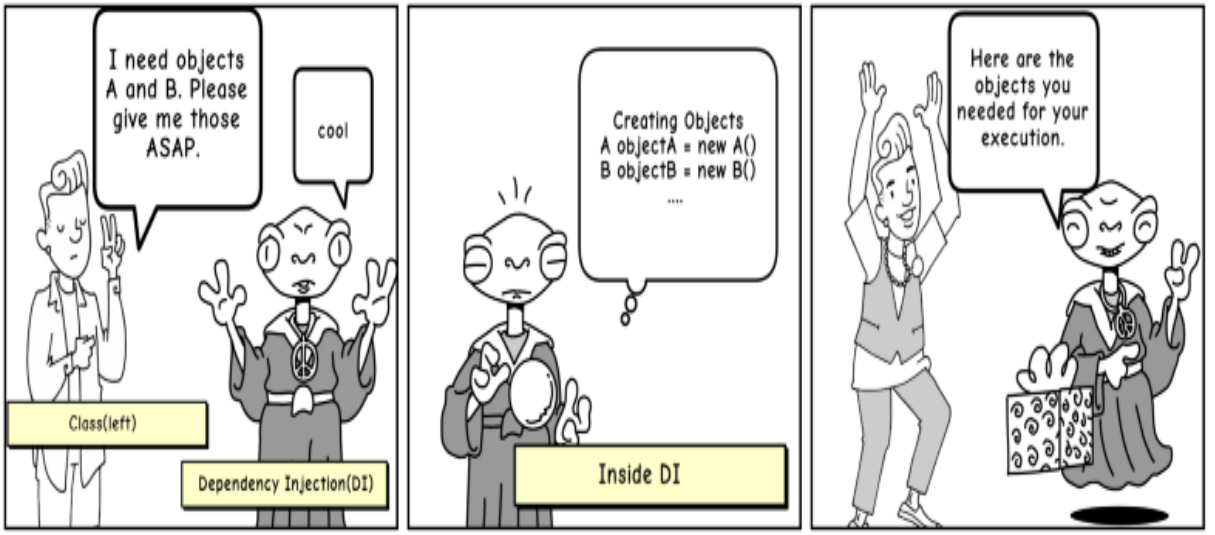
- **Spring MVC Request Flow:**

- DispatcherServlet receives HTTP Request.
- DispatcherServlet identifies the right Controller based on the URL.
- Controller executes Business Logic.
- Controller returns a) Model b) View Name Back to DispatcherServlet.
- DispatcherServlet identifies the correct view (ViewResolver).
- DispatcherServlet makes the model available to view and executes it.
- DispatcherServlet returns HTTP Response Back.

- **DispatcherServlet:**
- **View Resolver:**

2. Spring Framework - Java Spring the Modern Way:

- **Dependency injection:** transferring the task of creating the object to someone else and directly using the dependency is called dependency injection.



There are basically three types of dependency injection:

1. ***constructor injection***: the dependencies are provided through a class constructor.
2. ***setter injection***: the client exposes a setter method that the injector uses to inject the dependency.

3. ***interface injection***: the dependency provides an injector method that will inject the dependency into any client passed to it. Clients must implement an interface that exposes a setter method that accepts the dependency.

So now it's the dependency injection's responsibility to:
1. Create the objects
2. Know which classes require those objects
3. And provide them all those objects

If there is any change in objects, then DI looks into it and it should not concern the class using those objects. This way if the object's change in the future, then it's DI's responsibility to provide the appropriate objects to the class.

Inversion of control —the concept behind DI:

"This states that a class should not configure its dependencies statically but should be configured by some other class from outside."

Benefits of using DI
1. Helps in Unit testing.
2. Boilerplate code is reduced, as initializing of dependencies is done by the injector component.
3. Extending the application becomes easier.
4. Helps to enable loose coupling, which is important in application programming.

Refer:

https://www.freecodecamp.org/news/a-quick-intro-to-dependency-injection-what-it-is-and-when-to-use-it-7578c84fa88f/

- **CDI (Contexts and Dependency Injection)** is a standard dependency injection framework included in Java EE 6 and higher.

It allows us to manage the lifecycle of stateful components via domain-specific lifecycle contexts and inject components (services) into client objects in a type-safe way.

● Bean :

In short, a Spring bean is an object which Spring framework manages at runtime.

A Spring bean is a basic building block of any Spring application. Most of the application logic code you write will be placed in Spring beans.

There are **three different ways** in which you can define a Spring bean:

- **annotating** your class with the stereotype *@Component* annotation (or its derivatives)
- writing a bean factory method annotated with the *@Bean* annotation in a custom Java **configuration class**
- declaring a bean definition in an **XML configuration file**

Bean scope:

The scope of a Spring bean defines **how many instances of a particular class the framework creates at runtime**. The scope also describes on what condition a new object is created.

Spring offers several scopes for your beans. The core of the framework comes with two:

- singleton – a single instance
- prototype – multiple instances

Moreover, Spring comes with additional bean scopes dedicated to web applications:

- request
- session
- global session
- application

The default scope for all beans is the singleton. When a bean has the singleton scope, Spring creates only one instance and shares it across the whole application.

Use the ***@Scope*** annotation and its string attribute to select a scope.

for web scopes Spring comes with additional alias annotations:

- ***@RequestScope***
- ***@SessionScope***
- ***@ApplicationScope***

Bean initialization mode:

- Lazy
- Eager

*When your application starts, **Spring eagerly creates all singleton beans at the startup**. This default behavior allows us to quickly detect errors in bean definitions. On the other hand, eager **bean initialization makes the startup of your application slow**.So use **lazy initialization.***

Ex:
```
@Component
@Lazy
class MyLazyClass {
    //...
}
```

How does Spring create objects from bean definitions?

When you start a Spring application, the framework firstly creates a special object called ApplicationContext. The ApplicationContext, also known as the Inversion of Control (IoC) container, is **the heart of the framework**.

The **ApplicationContext is a container** in which your **bean objects exist.**

Refer:http://dolszewski.com/spring/spring-bean/

- **Autowire:**

3. Spring boot

"Convention over configuration"

1. What is Spring boot?

Easyway to create spring applications.

2. Why Spring boot?

*not much configuration.

*in built tomcat server, so no extra configuration required for deployment. Ready to run the application.

3. Spring boot features

SpringApplication.run(App.class, args) : this is responsible for:

*scans all the classes and checks what annotations are present and treats them accordingly.

*Runs the tomcat.

Log:

```
Searching directory
   [/in28Minutes/git/getting-started-in-5-steps/spring-in-5-steps/target/classes/com
   /in28minutes/spring/basics/springin5steps] for files matching pattern
   [/in28Minutes/git/getting-started-in-5-steps/spring-in-5-steps/target/classes/com
   /in28minutes/spring/basics/springin5steps/**/*.class]
Identified candidate component class: file
   [/in28Minutes/git/getting-started-in-5-steps/spring-in-5-steps/target/classes/com
   /in28minutes/spring/basics/springin5steps/BinarySearchImpl.class]
Identified candidate component class: file
   [/in28Minutes/git/getting-started-in-5-steps/spring-in-5-steps/target/classes/com
   /in28minutes/spring/basics/springin5steps/BubbleSortAlgorithm.class]

Creating instance of bean 'binarySearchImpl'
Creating instance of bean 'bubbleSortAlgorithm'
Finished creating instance of bean 'bubbleSortAlgorithm'

Constructor - Autowiring by type from bean name 'binarySearchImpl' via constructor
to bean named 'bubbleSortAlgorithm'
Setter -  Autowiring by type from bean name 'binarySearchImpl' to bean named
   'bubbleSortAlgorithm'
No Setter or Constructor - Autowiring by type from bean name 'binarySearchImpl' to
   bean named 'bubbleSortAlgorithm'


Finished creating instance of bean 'binarySearchImpl'
```

Refer:

https://github.com/PacktPublishing/Spring-Framework-Master-Class-Java-Spring-the-Modern-Way