Spring framework is [divided into modules](https://docs.spring.io/spring/docs/current/spring-framework-reference/index.html) which makes it really easy to pick and choose in parts to use in any application:

* [**Core**](https://docs.spring.io/spring/docs/5.1.8.RELEASE/spring-framework-reference/core.html#spring-core): Provides core features like DI (Dependency Injection), Internationalisation, Validation, and AOP (Aspect Oriented Programming)
* [**Data Access**](https://docs.spring.io/spring/docs/5.1.8.RELEASE/spring-framework-reference/data-access.html#spring-data-tier): Supports data access through JTA (Java Transaction API), JPA (Java Persistence API), and JDBC (Java Database Connectivity)
* [**Web**](https://docs.spring.io/spring/docs/5.1.8.RELEASE/spring-framework-reference/web.html#spring-web): Supports both Servlet API ([**Spring MVC**](https://docs.spring.io/spring/docs/5.1.8.RELEASE/spring-framework-reference/web.html#spring-web)) and of recently Reactive API ([**Spring WebFlux**](https://docs.spring.io/spring/docs/5.1.8.RELEASE/spring-framework-reference/web-reactive.html#spring-webflux)), and additionally supports WebSockets, STOMP, and WebClient
* [**Integration**](https://docs.spring.io/spring/docs/5.1.8.RELEASE/spring-framework-reference/integration.html#spring-integration): Supports integration to Enterprise Java through JMS (Java Message Service), JMX (Java Management Extension), and RMI (Remote Method Invocation)
* [**Testing**](https://docs.spring.io/spring/docs/5.1.8.RELEASE/spring-framework-reference/testing.html#testing): Wide support for unit and integration testing through Mock Objects, Test Fixtures, Context Management, and Caching.

**Inversion of Control:**

* Inversion of Control is a principle in software engineering which transfers the control of objects or portions of a program to a container or framework.
* **The spring container is at the core of the Spring Framework. The container will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction.**
* The main tasks performed by IOC container are:
* to instantiate the application class
* to configure the object
* to assemble the dependencies between the objects
* The container gets its instructions on what objects to instantiate, configure, and assemble by reading the configuration metadata provided. The configuration metadata can be represented either by XML, Java annotations, or Java code.
* The Spring IOC container makes use of Java POJO classes and configuration metadata to produce a fully configured and executable system or application.



* There are two types of IOC containers. They are:
* **BeanFactory**
* **ApplicationContext**
* The ApplicationContext interface is built on top of the BeanFactory interface.
* It adds some extra functionality than BeanFactory such as simple integration with spring's AOP, message resource handling, event propagation, application layer specific context (e.g. WebApplicationContext) for web application. So it is better to use ApplicationContext than BeanFactory.
* BeanFactory can still be used for lightweight applications like mobile devices or applet-based applications where data volume and speed is significant.

**Bean:**

* The objects that form the backbone of your application and that are managed by the Spring IOC container are called **beans**.
* A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IOC container.
* These beans are created with the configuration metadata that you supply to the container.
* When defining a bean you have the option of declaring a scope for that bean.
* The Spring Framework supports the following five scopes, three of which are available only if you use a web-aware ApplicationContext.

1. **Singleton:** If a scope is set to singleton, the Spring IOC container creates exactly one instance of the object defined by that bean definition. This single instance is stored in a cache of such singleton beans, and all subsequent requests and references for that named bean return the cached object.
2. **Prototype:** If the scope is set to prototype, the Spring IOC container creates a new bean instance of the object every time a request for that specific bean is made. **As a rule, use the prototype scope for all state-full beans and the singleton scope for stateless beans.**
3. **Request:** This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring Application Context
4. **Session:** This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.
5. **Global-session:** This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

Dependency Injection:

* Every Java-based application has a few objects that work together to present what the end-user sees as a working application.
* When writing a complex Java application, application classes should be as independent as possible of other Java classes to increase the possibility to reuse these classes and to test them independently of other classes while unit testing.
* **Dependency Injection (or sometime called wiring) helps in gluing these classes together and at the same time keeping them independent.**
* Connecting objects with other objects, or “injecting” objects into other objects, is done by an assembler rather than by the objects themselves.
* **Dependency Injection in spring can be done through constructors, setters or fields.**