Homework- 2

1- **Dependency injection (DI)** is the concept in which objects get other required objects from outside. Google guice is one of them for Java. Castle windsor,ninject,structure map,unity is dependency injection frameworks for .Net.

2- @EnableAutoConfiguration : Enables Spring Boot to auto-configure the application context. Therefore, it automatically creates and registers beans based on both the included jar files in the classpath and the beans defined by us.

@ComponentScan : Configures component scanning directives for use with @Configuration classes.Provides support parallel with Spring XML's <context:component-scan> element.

@SpringBootConfiguration : Indicates that a class provides Spring Boot application @Configuration. Can be used as an alternative to the Spring'sstandard @Configuration annotation so that configuration can be foundautomatically

@Retention : Indicates how long annotations with the annotated type are tobe retained

@Target : Indicates the contexts in which an annotation type is applicable

@Documented : It is a marker interface that tells a tool that an annotation is to be documented.

@Inherited : Indicates that an annotation type is automatically inherited.

3- @Primary : Indicates that a bean should be given preference when multiple candidates are qualified to autowire a single-valued dependency.
@Qualifier : It is used to resolve the autowiring conflict, when there are multiple beans of same type.

4- **Convention over configuration** is a software design paradigm used by software frameworks that attempts to decrease the number of decisions that a developer using the framework is required to make without necessarily losing flexibility and Don't repeat yourself (DRY) principles.
As an example, Maven uses Convention over Configuration, which means developers are not required to create build process themselves. Developers do not have to mention each and every configuration detail. Maven provides sensible default behavior for projects. When a Maven project is created, Maven creates default project structure.

5- **Aspect-oriented programming (AOP)** is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns.

Advantages :

- Complements object orientation.
- Modularizes cross-cutting concerns improving code maintainability and understandability.

Disadvantages :

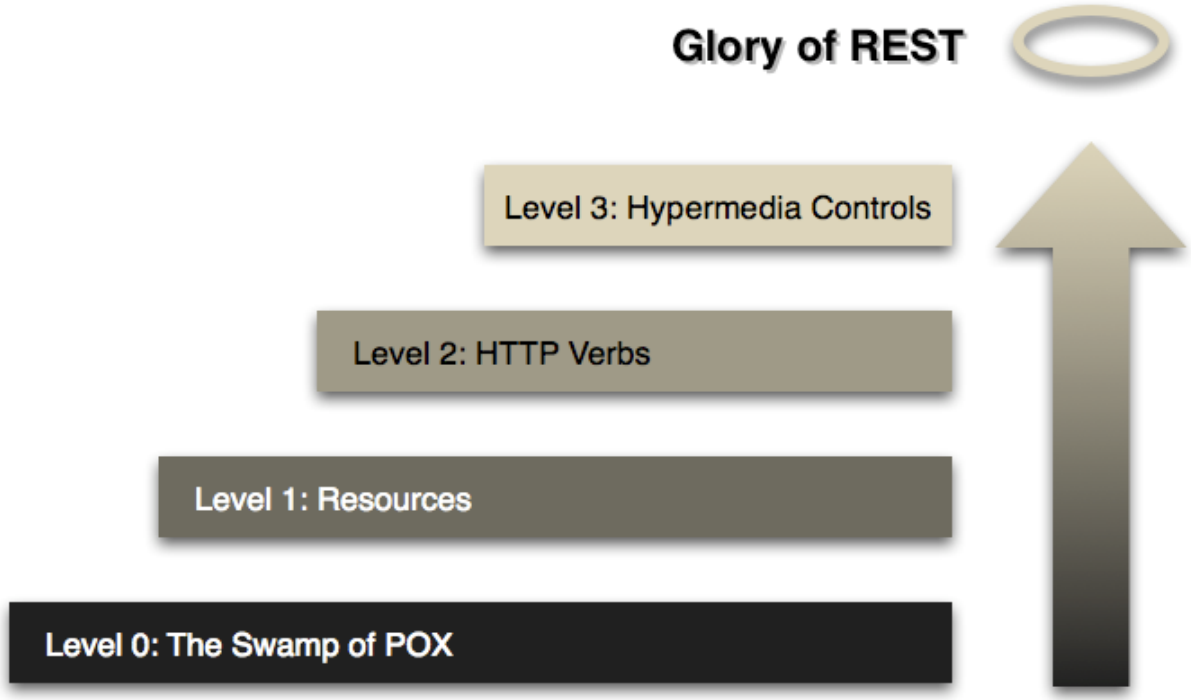
- It makes harder to testing and tracability

6- **S**ingle-Responsibility Principle : Modularizes cross-cutting concerns improving code maintainability and understandability.
Open-Closed Principle : Objects or entities should be open for extension but closed for modification.
Liskov Substitution Principle : Every subclass or derived class should be substitutable for their base or parent class.
Interface Segregation Principle : A client should never be forced to implement an interface that it doesn't use, or clients shouldn't be forced to depend on methods they do not use.
Dependency Inversion Principle : Entities must depend on abstractions, not on concretions. It states that the high-level module must not depend on the low-level module, but they should depend on abstractions.

We should develop our projects with solid principles as possible as. Because projects that adhere to SOLID principles can be shared with collaborators, extended, modified, tested, and refactored with fewer complications.

7- **Swagger** is an Interface Description Language for describing RESTful APIs expressed using JSON. Swagger is used together with a set of open-source software tools to design, build, document, and use RESTful web services. Swagger includes automated documentation, code generation (into many programming languages), and test-case generation.

8- **The Richardson Maturity Model (RMM)** is a maturity model suggested in 2008 by Leonard Richardson which classifies Web APIs based on their adherence and conformity to each of the model's four levels. The aim of the research of the model as stated by the author was to find out the relationship between the constraints of REST and other forms of web services.



Level 0 : The Swamp of POX : The lowest level of the model describes a Web API with a single URI (typically POST over HTTP) accepting all the range of operations supported by the service. Resources in this form cannot be well-defined. Messaging is done in XML, JSON, or other text formats. These are typical RPC POX and many SOAP services.

Level 1 : Resources : Introduces resources and allows to make requests to individual URIs (still all typically POST) for separate actions instead of exposing one universal endpoint (API). The API resources are still generalized but it is possible to identify the scope of each one.Level One design is not RESTful, yet it is organizing the API in the direction of becoming one.

Level 2 : HTTP Verbs : The system starts making use of HTTP Verbs. This allows to further specialize the resource and thus narrow down the functionality of each individual operation with the service. The principle separation at this level consists in splitting a given resource into two - one request for obtaining data only (GET), the other for modifying the data (POST). Further granularization is also possible. GET requests only fetch data, POST/PUT calls introduce new and update existing data, DELETE requests remove or otherwise invalidate previous actions. One drawback of providing a distributed service with more than GET and POST per resource might be growing complication of using such a system.

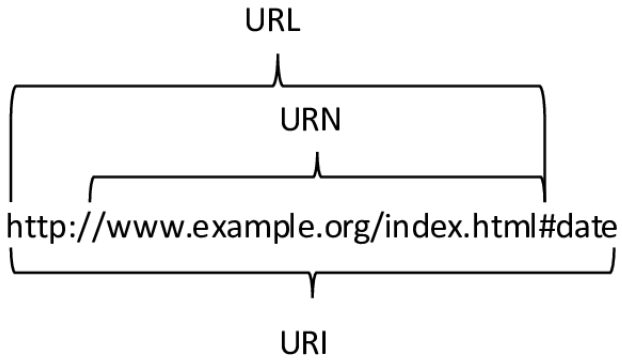
Level 3 : Hypermedia Controls : The last level introduces the hypermedia representation. Also called HATEOAS (Hypermedia As The Engine of Application State), these are elements embedded in the response messages of resources which allow to establish a relation between individual data entities returned from and pass to the APIs. For instance, a GET request to a hotel reservation system might return a number of available rooms along with hypermedia links (these would be html hyperlink controls in the early days of the model) allowing to book specific rooms.

9-

A **Uniform Resource Identifier (URI)** is a string of characters used to identify a name or a resource on the Internet.

A **Uniform Resource Locator (URL)** is a subset of the Uniform Resource Identifier (URI) that specifies where an identified resource is available and the mechanism for retrieving it.

A **Uniform Resource Name (URN)** is a Uniform Resource Identifier (URI) that uses the URN scheme, and does not imply availability of the identified resource.



10- **Idempotency** means that the result of a successfully performed request is independent of the number of times it is executed.

GET, **PUT**, **DELETE**, **HEAD**, **OPTIONS** and **TRACE** are idempotent in HTTP.

11- A **Request for Comments (RFC)** is a numbered document, which includes appraisals, descriptions and definitions of online protocols, concepts, methods and programmes.

RFC 7230 HTTP/1.1 : Message Syntax and Routing
RFC 7231 HTTP/1.1 : Semantics and Content
RFC 7232 HTTP/1.1 : Conditional Request
RFC 7233 HTTP/1.1 : Range Request
RFC 7234 HTTP/1.1 : Caching
RFC 7235 HTTP/1.1 : Authentication