```
Developing Spring boot application using Java Config Annotation
==============================================================
=> These are supplied by JSE,JEE modules(from java9 they are released as
independent libraries)
=> For these annotations, they underlying f/w or container or server decides the
functionality
      In Spring, the spring framework will decide the functionality.
      In hibernate, the hibernate framework will decide the functionality.
      In Servlet, the servlet container will decide the functionality.

eg: @PostConstruct,@PreDestroy,@Named,@Resource,@Inject,.....

@Named:: To configure java class as SpringBean and also to resolve ambiguity.
@Inject,@Resource :: They are alternative to @Autowired for Dependancy Injection.
                              @Resource can't be applied at constructor level
injection.

Note: Invasive(working to a company with bond) and Non-Invasive Programming(working
to a company without bond)

SpringBean class with Spring supplied annotations like
@Component,@Autowired,@Qualifier make spring bean class as "Invasive".
To make SpringBean class as non-invasive take the support of Java Config
Annotation.

Note:As of now very limited java config are available,so it is practially
impossible to develop entire spring or
     spring boot application using java config annotation.
       so we prefer the following order
            a. Java config annotation
            b. Spring annotations
            c. third party annotations
            d. custom annotations

To use java config annotations we need to add the following jar file

pom.xml
=======
<dependency>
      <groupId>javax.inject</groupId>
      <artifactId>javax.inject</artifactId>
      <version>1</version>
</dependency>

=> @Inject can be used at field level,construtor level,setter method level
=> @Resource can be used at field level,setter method level.
=> While working with @Inject we need to use another annotation called @Named to
resolve the ambiguity problem.
=> While working with @Resource only "name" param itself would resolve the problem.

                                    refer:: BootProj07-DependancyInjection-
JavaConfiguration



Note:
As of Springboot2.5+ version is using two Datasources as a part of
AutoConfiguration if we add spring-boot-starter-jdbc
      a. hikari cp(default)
      b. Apache dbcp2 datasource(only when hikaricp jars are not there in the
classpath)
      c. tomcat-dbc(only when hikaricp jars are not there in the classpath)

Priority order for AutoConfiguration is
      a. hikaricp(best choice)
      b. tomcat
      c. dbcp2

DBCP2
=====
<dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-dbcp2</artifactId>
</dependency>

tomcat
======
<!-- https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-jdbc -->
<dependency>
      <groupId>org.apache.tomcat</groupId>
      <artifactId>tomcat-jdbc</artifactId>
</dependency>


How can we make dbcp2 datasource to work with SpringBoot?
 => exclude hikaricp jar file from dependent jar of "spring-boot-starter-jdbc"

pom.xml
=======
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-jdbc</artifactId>
      <exclusions>
            <exclusion>
                  <groupId>com.zaxxer</groupId>
                  <artifactId>HikariCP</artifactId>  ====> go to dependancy
hierarchy tab(right click on hikaricp ,exclude maven artifact)
            </exclusion>
      </exclusions>
</dependency>

Add apachedbcp2 jar files

pom.xml
=======
<dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-dbcp2</artifactId>
</dependency>

Can we disable autoconfiguration of certain spring bean even though starters are
added?
Ans. yes,we can do by using using exclude param
            @SpringBootApplication(exclude = { DataSourceAutoConfiguration.class,
JdbcTemplateAutoConfiguration.class })
            public class BootProj06ConfigurationPropertiesAppApplication {
                  public static void main(String[] args) {
                                          ;;;;;
                  }
            }
```
In the above case we need to use @Bean method to create ur choice class objects and to make them as spring bean either in @Configuration
class or in @SpringBootApplication class.

```
@Configuration
public class PersistConfig {

      @Autowired
      private Environment env;

      @Bean
      public ComboPooledDataSource createDS() throws Exception {
            System.out.println("PersistConfig.createDS()");
            ComboPooledDataSource dataSource = new ComboPooledDataSource();
            dataSource.setJdbcUrl(env.getProperty("spring.datasource.url"));
            dataSource.setUser(env.getProperty("spring.datasource.username"));
            dataSource.setPassword(env.getProperty("spring.datasource.password"));
            return dataSource;
      }
}
```

application.yml
===============
```
spring:
  datasource:
    password: root123
    url: jdbc:mysql:///enterprisejavabatch
    username: root
```

pom.xml
======
```
<dependency>
      <groupId>com.mchange</groupId>
      <artifactId>c3p0</artifactId>
      <version>0.9.5.4</version>
</dependency>
```

EmployeeDaoImpl.java
====================
```
@Repository
public class EmployeeDaoImpl implements IEmployeeDAO {

      private static final String SQL_SELECT_QUERY = "select
eid,ename,eage,eaddress from employee";

      @Autowired
      private DataSource dataSource;

      @Override
      public List<Employee> findAllEmployees() throws Exception {
            System.out.println("DataSource Connection is :: " +
dataSource.getClass().getName());
      }
}
```

Output

```
      DataSource Connection is :: com.mchange.v2.c3p0.ComboPooledDataSource


Usage of SPEL in realtime
=========================
BillGenerator.java
==================
@Component("bill")
public class BillGenerator {

      @Value("#{item.idlyPrice+item.dosaPrice+item.vadaPrice}")//SPEL => used for
computation
      private Float billAmount;

      @Value("Accord")
      private String hotelName;

      @Autowired
      private ItemsInfo info;

      @Override
      public String toString() {
            return "BillGenerator [billAmount=" + billAmount + ", hotelName=" +
hotelName + ", info=" + info + "]";
      }
}


ItemsInfo.java
==============
@Component("item")
public class ItemsInfo {

      @Value("${items.info.idlyPrice}")
      public float idlyPrice;

      @Value("${items.info.dosaPrice}")
      public float dosaPrice;

      @Value("${items.info.vadaPrice}")
      public float vadaPrice;

      @Override
      public String toString() {
            return "ItemsInfo [idlyPrice=" + idlyPrice + ", dosaPrice=" + dosaPrice
+ ", vadaPrice=" + vadaPrice + "]";
      }

}

application.properties
======================
items.info.idlyPrice= 10
items.info.dosaPrice= 20
items.info.vadaPrice= 30


BootProj07DependancyInjectionJavaConfigurationApplication.java
==============================================================
@SpringBootApplication
public class BootProj07DependancyInjectionJavaConfigurationApplication {

      public static void main(String[] args) throws Exception {
            ApplicationContext context = SpringApplication
                        .run(BootProj07DependancyInjectionJavaConfigurationApplicat
ion.class, args);

            System.out.println("Beans info are :: " +
Arrays.toString(context.getBeanDefinitionNames()));
            System.out.println();

            BillGenerator billGenerator = context.getBean(BillGenerator.class);
            System.out.println(billGenerator);

            ((ConfigurableApplicationContext) context).close();
      }
}
```

output

```
BillGenerator [billAmount=60.0, hotelName=Accord, info=ItemsInfo [idlyPrice=10.0,
dosaPrice=20.0, vadaPrice=30.0]]
```