```
Lombok api, eclipse debugging ===> session on monday from 7.30PMIST to 11.00PM IST
link will be sent to both the batches


Spring Data
===========
 => Spring data JPA
 => Spring data MongoDB


Before the arrival of Spring data module
========================================
  SpringAPP ====Spring JDBC============> RDMB s/w or SQL DBS/W
                (JDBCstyle)

  SpringApp =====Spring ORM============> RDBMS s/w or SQL DBS/W
                 (OR-Mapping)

  SpringApp ====MongoDB API + MongoDB Driver======>MongoDB S/W(NOSQL DBS/w)

  SpringApp ====Casendra API + Casendra Driver======>Cassendra S/w(NOSQL DBS/w)


Note: Spring is not having any module to interact with NO SQL DBs/w before the
arrival of Spring Data module
Note: Before arrival of Spring data module there is no single unified mechanism to
talk both SQL and NoSQL DB s/w from Spring.
      We need to use different types of SQL or NO-SQL DBS/W.

SpringData module provides abstraction on multiple technologies and frameworks to
simplify the interaction b/w both SQL and
NO SQLDBs/w in the unified model environment.


Important sub modules of spring data module
===========================================
 a. Spring data JPA     => provides abstraction on ORM S/w
 b. Spring data JDBC    => provides abstraction on JDBC technology.
 c. Spring data MongoDB => provides abstraction on MongoDB api and etc....


Main Modules
============
  a. Spring Data JPA     => Spring Data Repository support for JPA
  b. Spring Data MongoDB => Spring based, object document support and repositories
from MongoDB.


Limiation of SpringORM
======================
  1. If we have 500 tables, then we need to have 500 DAO interfaces, followed by
500 implementation classes for all the interface
     and in all the implementation class the CRUD operation would be a common
operation.
  2. In the above approach, the logic of many operations would be duplicated
resulting in "boiler plate" code.
  3. To resolve this problem Spring community had come up with a module called
SpringDATA which would generate redundant code
     automatically through an Pre-Defined Repositories.
```

```
Spring Data JPA code
====================
  => Just create Repository/DAO Interface extending        PreDefined Repository 
Interface
                                                                       (different 
types of repository interfaces are there)

      (logic of many method prewritten)

  => If needed declare some custom methods by following coding conventions.

  Advantages with SpringDataJPA
  =============================
=> Now for 500 Db Tables, we just need to take 500 CustomRespository interfaces 
having optional custom delare methods.
=> Implementation classes of CustomRespository interface will be generated 
dynamically providing persistence logics for common 
   methods which are inherited from Pre-Defined Repository interface and also for 
custom method declartion.
     Note: All these operations are taken care by springdatajpa using InMemory 
Proxy-classes.


Normal Java class
=================
   .java =====> .class(HDD) ======> JVM loads the .class file(JVM memory RAM) ===> 
execution class file(JVM Memory of RAM)


In memory java class
====================
   Run the application ===> source code generation(JVM memory RAM) ===> 
compilation(JVM memory of RAM)
            ===> JVM loads the .class file =====> execution of class file(JVM 
memory of RAM)

Note:: Spring DataJPA uses ProxyDP to generate the implementation class of 
programmer supplied DAO/Repository interface as
       a InMemory Proxy class dynamically at the runtime.
         While working with Spring DataJPA, the peristence layer just contains 
DAO /Repository interface having just few 
       custom method declarations.


Spring Data JPA 
  => It internally uses hibernate as ORM Framework
  => Strong knowledge of hibernate is required to understand.
  => While working with annotations we need to prefere the following order to build 
Entity classes
                  a. JPA annotation
                  b. Java config annotations
                  c. hibernate specific annotations
                  d. Third party supplied annotations


Repositories
```

```
============
  1.Repository
  2.CrudRepository
  3.PagingAndSortingRepository
  4.JpaRepository

ids:: 1,2,4,5 ===> deletion should be succesfull
ids:: 3,5,6   ===> problem in deletion
                  refer:: DAO-SpringDataJPA-CRUDRepository



PagingAndSortingRepository(I)
   => It is given to perform sorting of records and pagination.
   => Pagination will display huge amount of records page by page.
   => We can sort the records either in Ascending order or in Descending order
            In Ascending order
                  a. special characters(*,?,-,...)
                  b. numbers(0-9)
                  c. upper case alphabets(A-Z)
                  d. lower case alphabets(a-z)


Iterable<T> findAll(Sort sort)
==============================
 => It performs sorting in ascending order or descending order based on the given
property(single/mulitple)
 => We need to specify the inputs to sort through Sort Object.


Page<T> findAll(Pageable pageable)
==================================
 => This method takes pageNo(0 based),pageSize as the input in the form of Pageable
object and returns the output as Page<T> obj
    having requested pageRecords,pagesCount,currentnumbers,total records etc.....

total records  => 20
page size      => 5
   pages       => 4
   pageNo      => 0,1,2,3[Every page 5 records so totally 20 records]
======================================================================
total records  => 20
page size      => 5
    pages      => 3
    pageNo     => 0,1,2[Every page 5 records, autoamtically 3th page 16-20 records]
========================================================================
total records  => 20
page size      => 5
    pages      => 2
      pageNo     => 0,1[Every page 5 records,automatically 2nd page 11 to 15
records,autoamtically 3th page 16-20 records]


                        refer:: DAO-SpringDataJPA-PagingAndSortingRepository
```